ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

Autentificación y firma digital mediante DNI electrónico

Xabier Esteban Lozano

Mikel Izal Azcárate

Pamplona, 14/9/2010

1. Introducción

En este proyecto se pretenden investigar las librerías y API's que permiten utilizar las funciones del DNI electrónico y utilizarlas en aplicaciones de ejemplo construyendo programas o páginas web que utilicen autentificación mediante el DNI electrónico o firma electrónica basada en el DNIe.

2. Antecedentes y objetivos

Para poder realizar el proyecto es necesario comprender perfectamente cómo funciona el DNI electrónico. Para ello será imprescindible conocer conceptos de criptografía tales como la autentificación, la firma digital o el cifrado. Tras estudiar estas ideas será el momento de construir unos ejemplos que nos permitan mostrar el uso del DNI electrónico y comprender mejor cómo funciona. En este caso realizaremos 2 ejemplos:

Un chat en el que los usuarios se autentificarán mediante el DNIe y podrán intercambiar mensajes con otros usuarios de forma segura y cifrada. Esto nos permitirá conocer la API de funciones del DNIe, profundizar en el uso de SSL y comprender mejor cómo funciona el cifrado de texto y la autentificación de usuarios.

Una página web en la que los clientes se autentificarán mediante el DNIe y se les mostrará una información personalizada. Además tendrán la posibilidad de firmar documentos en línea, todo ello de forma segura y con validez legal. Así conseguiríamos profundizar en el concepto de firma digital; además mediante el protocolo https veríamos un ejemplo de autentificación y cifrado de mensajes muy extendido y que nos permitirá actuar con el DNIe a través del navegador web.

3. Desarrollo del trabajo

3.1. Requisitos

Para la utilización de nuestro DNIe necesitaremos tener instalados en nuestro sistema una serie de requisitos tanto de hardware como de software.

En cuanto al hardware, necesitaremos a parte de un ordenador personal, un lector de tarjetas inteligentes que cumpla el estándar ISO-7816, bien sea integrado en el teclado, bien externos USB o a través de una interfaz PCMCIA. Para que un lector sea compatible con el DNI electrónico debe cumplir:

- El estándar ISO-7816 (1, 2 y 3)
- Soportar tarjetas asíncronas basadas en protocolos T=0 (y T=1)
- Soporte para velocidades mínimas de comunicación de 9600 bps
- Y soporte para los siguientes estándares:
    - API PC/SC (Personal Computer/Smart Card)
    - CSP (Cryptographic Service Provider, Microsoft)
    - API PKCS#11

Todos estos requisitos son cumplidos por el lector C3PO LTC31, un lector externo (USB) de reducidas dimensiones que será el que utilizaremos durante todo el proyecto.

En cuanto a elementos software, el DNI electrónico es compatible con diferentes sistemas operativos (Windows, Linux, Unix y Mac). En este caso el sistema operativo que vamos a utilizar será Linux ya que su distribución bajo licencia GNU hace gratuita su instalación en cualquier equipo.

El DNIe se puede integrar en los navegadores Internet Explorer, Mozilla Firefox y Netscape. En nuestro caso hemos elegido el navegador Mozilla Firefox (versión 1.5 o superior) al ser el más extendido en los sistemas GNU/LINUX.

Además de un sistema operativo y un navegador compatible, necesitaremos unos controladores o módulos criptográficos, para poder interaccionar con las tarjetas. En entornos Windows es necesario tener instalado un servido llamado CSP (Cryptographic Service Provider) mientras que para sistemas UNIX, Linux o Mac utilizaremos nuestro DNIe a través del Modulo PKCS#11. Nosotros lógicamente utilizarnos este último. Ambos módulos pueden obtenerse desde la sección de descargas de la página oficial del DNI electrónico: www.dnielectronico.es/descargas

Ahora que ya conocemos los requisitos para poder utilizar nuestro DNIe podemos comenzar a instalarlos.

3.2. Instalación

Lógicamente, lo primero que debemos instalar en nuestro equipo es un sistema operativo. Como hemos dicho, vamos a instalar un sistema operativo GNU/LINUX, debido a su distribución libre, por lo que podemos instalarlo en cualquier equipo sin coste alguno. Además es una muy buena opción para el desarrollo de aplicaciones debido al compilador gcc. Linux nos permitirá también, desarrollar y mantener nuestra página web con programas muy extendidos como apache o mySQL server.

Dentro de las muchas distribuciones de GNU/Linux, hemos elegido la que probablemente esté a día de hoy más extendida entre los usuarios de este sistema operativo: Ubuntu, y más concretamente su versión estable 9.10. Pese a haber elegido esta distribución en concreto, se podría realizar el proyecto en otras muchas distribuciones sin mayor complicación.

La instalación de este sistema operativo se sale un poco del objetivo de este proyecto, aunque a día de hoy instalar este tipo de sistemas es bastante intuitivo y existe numerosa documentación sobre este tema en la web.

Una vez tenemos nuestro sistema operativo instalado, es la hora de instalar los controladores de nuestro lector LTC31. Para ello deberemos instalar los siguientes paquetes:

- Libccid: Driver genérico CCID del lector de tarjetas chip.
- Libpcsclite: Librerías del sistema para comunicarse con el lector.
- Pcscd: Servicio de tarjeta inteligente para comunicarse con las librerías anteriores.
- Libusb: librerías para comunicarse con dispositivos USB.

Para instalar estos paquetes tenemos dos opciones: instalarlos de manera gráfica a través del gestor de paquetes Synaptic, o bien a través de consola, buscando los paquetes con el comando apt-cache search nombre del paquete e instalándolos después mediante el comando apt-get install nombre del paquete.

Una vez hecho esto ya tendremos configurando correctamente nuestro lector, por lo que ahora tendremos que instalar los controladores necesarios para que nuestro navegador Mozilla Firefox interactúe con nuestro DNIe. El navegador Mozilla Firefox suele estar instalado por defecto en muchas distribuciones de Linux, pero de no ser así, podemos instalarlo de manera análoga a las librerías que acabamos de instalar.

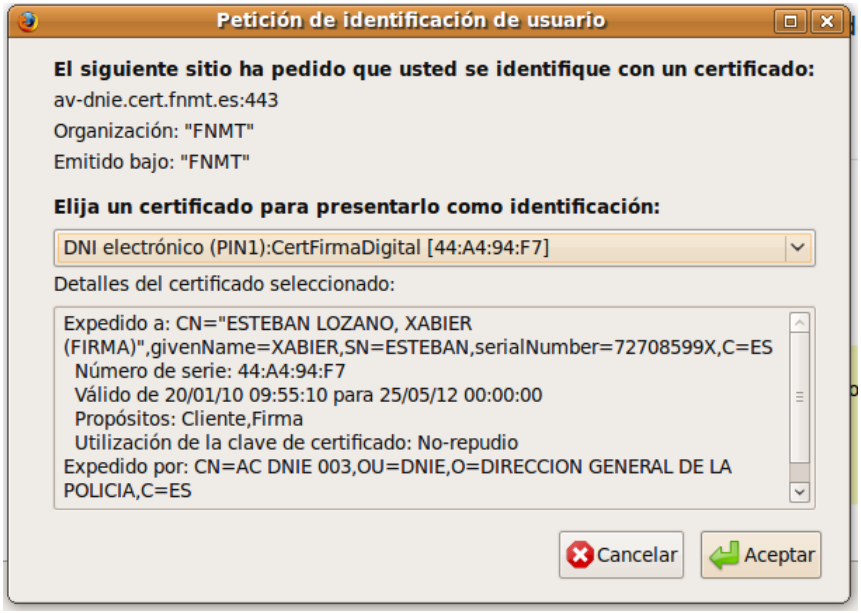
Vamos al área de descarga de la web oficial del DNI electrónico y de ahí descargamos el comprimido opensc-dni para nuestra distribución Ubuntu y para nuestra arquitectura, en este caso 32 bits. El comprimido contendrá los siguientes paquetes .deb que deberemos instalar:

- Libopensc2_0.11.7-7_i386.deb
- Opensc_0.11.7_7_i386.deb
- Opensc-dnie_1.4.6-2_i386.deb

Para instalar estos paquetes podemos hacerlo a través de la consola con el comando dpkg –i o bien, simplemente dando doble click sobre el paquete y posteriormente pulsando el botón instalar paquete. Tras instalar estos paquetes se le avisará al usuario que debe configurar su navegador para poder hacer uso de su DNIe. Esto se puede hacer automáticamente a través del menú Aplicaciones / Oficina / Registrar Modulo del DNIe PKCS#11 o bien manualmente a través del Firefox. Para ello iremos a Preferencias / Avanzado / Cifrado / Dispositivos de seguridad y cargar el modulo /usr/lib/opensc-pkcs11.so.

Con todo configurado no nos queda más que probar que verdaderamente funciona. Para ello, la página oficial del DNI electrónico tiene un enlace para verificar que todo está correctamente configurado. Así pues, nos dirigimos a la dirección:
https://av-dnie.cert.fnmt.es/compruebacert/compruebacert

Con el DNIe en nuestro lector, el navegador nos pedirá el PIN de nuestro DNI, y a continuación se mostrará un menú para elegir que certificado seleccionar.



Elegimos el de autentificación y si todo está correctamente visualizaremos algo parecido a esto:

COMPROBACIÓN DEL CERTIFICADO DE AUTENTICACIÓN DE SU DNI ELECTRÓNICO

Estimado Sr/Sra. ############

Su DNIe acaba de ser verificado. Esta usted en disposición de un Certificado de **Autenticación Activo**.

Identificador	Valor
INFORMACIÓN SOBRE LA IDENTIDAD	**(Valores Personales)**
Nombre	###### (AUTENTICACIÓN)
Apellidos	############
NIF	#########
Número de Serie del Certificado de Autenticación	############ ################## ##################
Autoridad Emisora	AC DNIE 002
Propietario	########## ################ ################
Comienzo de la Validez del Certificado	##########
Fin de la Validez del Certificado	25 de septiembre de 2010
Estado del Certificado de Autenticación	Activo

Veremos una página en la que se muestran nuestros datos, además de un campo de texto en el que podremos introducir una frase y firmarla. El proceso para firmar es el parecido. Se nos pedirá que seleccionemos el certificado de firma y que introduzcamos el código Pin.

Tras realizar la firma se nos mandará a una página diciendo que todo funciona correctamente.

Si no hemos podido visualizar la página, puede haber dos posibles problemas: que no hayamos instalado todo correctamente, o que los certificados del DNIe estén revocados. En este caso, la solución sería subir a una comisaría de policía y renovar los certificados.

Ya hemos instalado correctamente nuestro DNIe y hemos visto que funciona, pero todavía no sabemos nada de para qué sirve realmente un DNIe y cómo funciona internamente.

3.3. EL DNIe. ¿Qué es y qué ventajas tiente?

El documento nacional de identidad emitido por la dirección general de la policía nacional acredita desde hace más de 50 años, la identidad, los datos personales y la nacionalidad española de su titular.

Con la llegada de las nuevas tecnologías y la generalización del uso de internet, ha sido necesaria la evolución del carnet de identidad para trasladar al mundo digital las mismas certezas con las que operarnos a diario en el mundo físico. Esencialmente son:

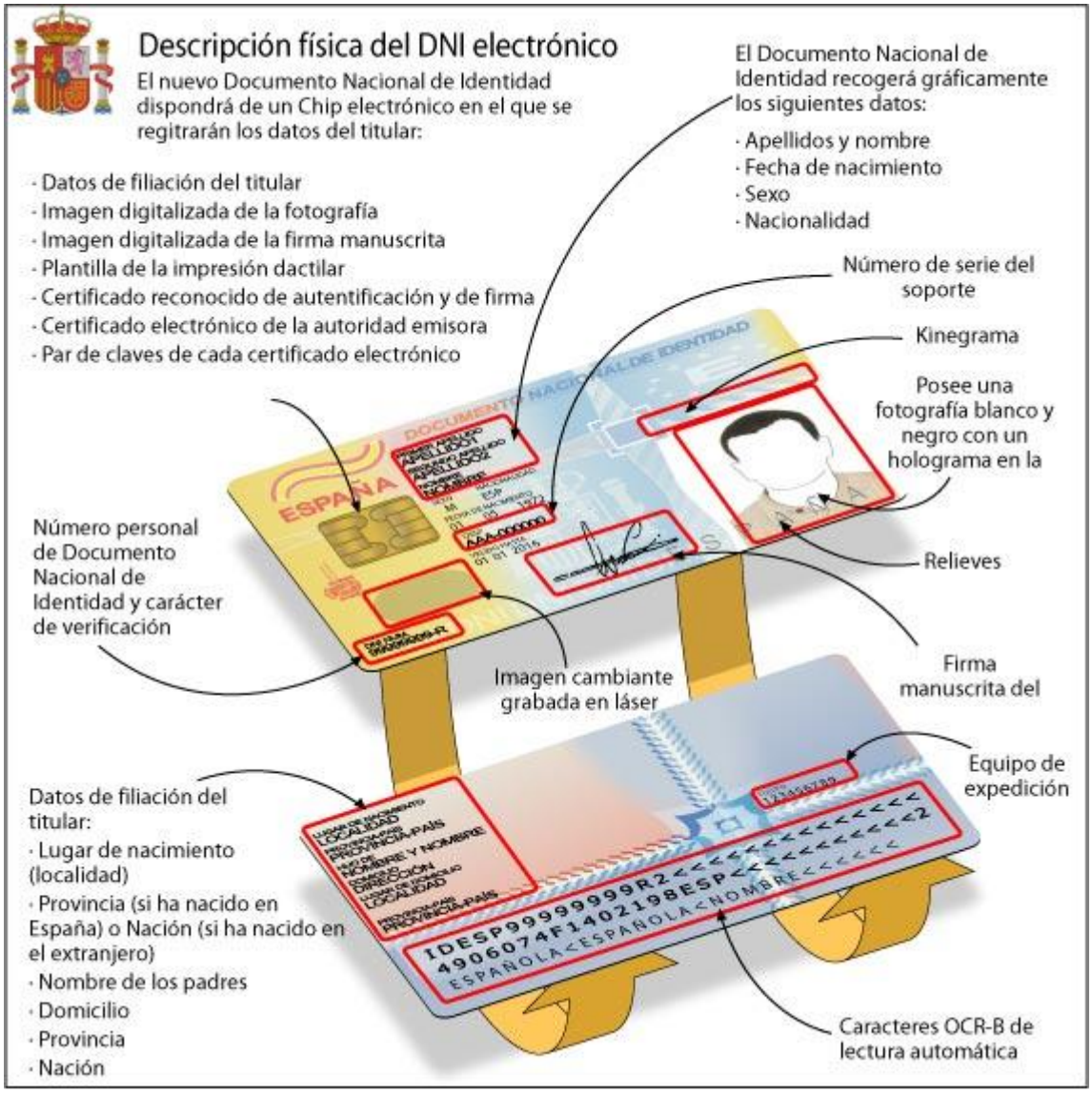
- Acreditar individualmente y de forma digital la identidad de una persona.
- Firmar digitalmente documentos electrónicos, con la misma validez jurídica que tendría una firma manuscrita.

Para ello se ha creado el nuevo DNI electrónico, similar al anterior, pero que incorpora un circuito integrado, capaz de guardar y procesar información de forma segura. Por este motivo se ha cambiado el soporte tradicional, por una tarjeta de plástico con mayores medidas de seguridad. Además de su uso tradicional este nuevo DNIe nos permitirá actuar a distancia con las administraciones públicas, con las empresas e incluso con otros ciudadanos.

En la medida en que el nuevo DNIe vaya sustituyendo al tradicional y se implanten las nuevas aplicaciones podremos:

- Realizar compras firmadas a través de internet.
- Hacer trámites completos con las administraciones públicas, en cualquier momento del año y sin la necesidad de hacer colas.
- Transacciones seguras con entidades bancarias.
- Acceder al edificio donde trabajamos.
- Utilizar de forma segura nuestro ordenador personal.
- Participar en una conversación por internet, con la certeza de que el interlocutor es realmente quien dice ser.

3.4. DNIe. Descripción



En el anverso de la tarjeta se encuentran los siguientes elementos:

- Primer Apellido
- Segundo Apellido
- Nombre
- Sexo y Nacionalidad
- Fecha de Nacimiento
- IDESP: Numero de serie del soporte físico de la tarjeta.
- Fecha de validez

- DNI num: Número del documento nacional de identidad del ciudadano, seguido del carácter de verificación.
- Fecha de expedición
- Primera consonante de cada apellido + Primera consonante del nombre
- Chip criptográfico:
    - Certificado electrónico para autentificar la identidad del ciudadano.
    - Certificado electrónico para firmar documentos con la misma validez jurídica que una firma manuscrita.
    - Certificado de la autoridad de certificación emisora.
    - Claves para su utilización.
    - Plantilla biométrica de la impresión dactilar.
    - Imagen digitalizada de nuestra firma manuscrita.
    - Datos de filiación del ciudadano, correspondientes con el contenido personalizado en la tarjeta.
- Elementos de seguridad:
    - Medidas de seguridad físicas:
        - Visibles (tintas ópticamente variables, relieves, fondos de seguridad)
        - Tintas visibles con luz ultravioleta, micro escrituras.
    - Medidas de seguridad digitales:
        - Encriptación de los datos del chip.
        - Acceso a la funcionalidad del DNIe mediante la clave Pin.
        - Las claves nunca abandonan el chip.
        - La autoridad de certificación es la Dirección general de la policía.

En el reverso de la tarjeta:

- Lugar de nacimiento
- País-Provincia
- Hijo de
- Domicilio
- Lugar de domicilio
- Provincia País y equipo
- Información impresa OCR-B para lectura mecanizada.

El DNIe no contiene ninguna otra información relativa a datos personales ni de cualquier otro tipo (sanitarios, fiscales, tráfico).

En caso de pérdida o sustracción del documento se deberá denunciar, para así poder revocar los certificados, tanto de autentificación como de firma digital. La custodia de las claves privadas la realizan los ciudadanos y en ningún caso la administración guarda copia de las mismas. Es decir, las claves privadas nunca abandona la tarjeta, minimizando la probabilidad de poner en riesgo esas claves.

Para acceder a los certificados o claves, se emplea un código de acceso PIN generado en el momento de recibir su DNIe. Se podrá modificar este PIN de forma telemática siempre y cuando se recuerde el código pin antiguo. De no ser así, o en caso de bloqueo (3 fallos), se podrá realizar el cambio mediante los puestos destinados para ello en las comisarías de policía mediante la comprobación de la biometría de la huella dactilar.

Dada la importancia de este código es importante tener en cuenta que no es muy seguro apuntarlo, enviarlo, ni comunicárselo a nadie por ningún medio. Además es recomendable que no sea un código relacionado con tus datos personales y por supuesto es una buena práctica cambiarlo cada cierto tiempo.

3.5. DNIe Contenido del chip

Como hemos visto, la principal novedad que posee el DNIe es la presencia de un chip ST19WL34, que tiene una capacidad de 32K. Este chip está dividido en tres zonas de diferentes niveles y accesos.

- Zona pública: Accesible en lectura sin restricciones.
    - Certificado CA intermedia emisora
    - Claves Diffie-hellman
    - Certificado x509 de componente
- Zona privada: Accesible en lectura mediante el uso del código Pin
    - Certificado de firma (no repudio)
    - Certificado de autentificación (Digital signature)
- Zona de seguridad: Accesible en lectura, sólo en los puntos de actualización del DNIe.
    - Datos de filiación del ciudadano
    - Imagen de la fotografía
    - Imagen de la firma manuscrita.

Es decir, el DNIe posee: claves públicas y privadas, tanto para la autentificación como para la firma, así como claves de la Autoridad certificadora. Junto a estas claves tenemos sus correspondientes certificados:

- Certificado de componente: Cuyo propósito es la autentificación de la tarjeta del DNIe y que permite el establecimiento de un canal cifrado entre la tarjeta y los drivers.
- Certificado de autentificación: Tiene como finalidad garantizar la identidad del ciudadano al realizar una operación telemática. El titular podrá garantizar su identidad acreditando la posesión tanto del certificado como de su clave asociada.
- Certificado de firma: Que utilizaremos para firmar documentos garantizando la integridad del documento y el no repudio de origen. Este certificado es expedido como certificado reconocido y creado en un dispositivo seguro de creación de firma, lo que convierte la firma electrónica avanzada, en firma

electrónica reconocida, permitiendo su equiparación legal con la firma manuscrita.

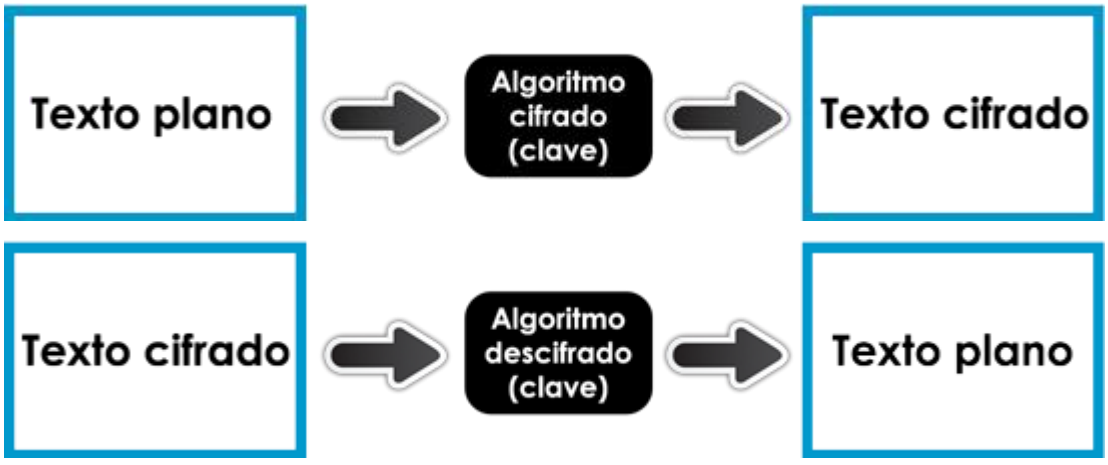
A grandes rasgos, ya hemos visto qué es lo que se puede hacer con un DNIe, pero seguimos sin saber realmente cómo funcionan y muchos términos pueden resultarnos confusos o desconocidos: autentificación, firma avanzada, firma reconocida, cifrado, autoridad certificadora, clave pública y privada… Por esto es imprescindible aprender algunas de las ideas básicas de la criptografía.

3.6. Criptografía

Es la práctica y el estudio de la ocultación de la información. Es decir, es el arte de cifrar o descifrar información. Se emplea para intercambiar mensajes que sólo pueden ser leídos por las personas a los que van dirigidos.

La criptografía que usamos y conocemos data de los años 70, aunque ya se usaban técnicas criptográficas durante la segunda guerra mundial, o incluso se conoce su uso desde la época de los griegos o los romanos. A día de hoy, existen multitud de canales no seguros que se basan en la criptografía para aportar mayor seguridad. La consolidación de estándares de firma digital y en particular los relativos a la infraestructura PKI han permitido una facilidad de uso que antes no existía, lo que ha permitido a las empresas acceder a ellas y aprovecharse de las ventajas que ofrece. Por este motivo, infinidad de usos se aplican cada día en el mundo de la tecnología, ya sea en comunicaciones militares envíos de correos electrónicos cifrados, protocolo https o cualquier otra comunicación telemática.

El cifrado es el proceso de convertir el texto plano (así llamamos al texto original) en un texto imposible de leer llamado texto cifrado. La idea más básica es: Para el proceso de cifrado se emplea un algoritmo de cifrado, que utiliza una clave. De forma análoga, si queremos descifrar el texto, utilizaremos la misma clave con un algoritmo de descifrado. Lógicamente, esta clave la deben conocer las dos partes, pero nadie más, si no la integridad del texto se vería comprometida.



Dependiendo del tipo de clave a utilizar existen dos tipos de sistemas criptográficos: criptografía simétrica y criptografía asimétrica.

3.6.1. Criptografía simétrica

En este sistema se utiliza una única clave para cifrar y descifrar los mensajes. Las dos partes deben ponerse de acuerdo sobre que clave usar y una vez hecho, tanto el cifrado como el descifrado de los mensajes se realizará con esa clave.

Actualmente los ordenadores pueden descifrar claves con extrema rapidez, por este motivo, el tamaño de la clave es muy importante. Por ejemplo el algoritmo DES utiliza una clave de 56 bits, lo que supondría un total de 2 elevado a 56 claves posibles. Pese a ser un numero alto, un ordenador moderno podría comprobarlas en cuestión de días y uno especializado sólo en unas horas. Algoritmos más recientes usan claves de entre 128 y 256 bits.

Sin embargo, el mayor problema de este tipo de criptografía radica en el intercambio de claves. Ningún canal de comunicación es lo suficientemente seguro como para garantizar que nadie interceptará dicha clave. A esto hay que unir otro grave problema, el número de claves necesarias. Para que n personas se comunique entre sí, serían necesarias n-1 claves por cada persona, es decir n*((n-1)/2). Para un grupo pequeño de personas sería factible, para 4 personas se necesitarían 6 claves; sin embargo este sistema se vuelve inviable para un numero alto de personas, ya que para 100 personas necesitaríamos la friolera de 4950 claves. Dados estos problemas, es obvio que necesitamos una solución mejor, la criptografía asimétrica.

3.6.2. Criptografía asimétrica

Este sistema usa dos claves para el envió de mensajes: la clave pública y la clave privada. Cada usuario posee una cave pública y una privada. Se debe mantener en secreto la privada y compartir la pública con los receptores con los que se desea comunicarse.

Los métodos criptográficos garantizan que sólo se pueden generar esa pareja de claves una vez, por lo que no es posible que dos personas hayan obtenido la misma pareja de claves.

En este sistema, lo que cifra la clave pública sólo se puede descifrar con su clave privada correspondiente, y lo que cifra la clave privada sólo se puede descifrar con su clave pública correspondiente.

El procedimiento es bien sencillo, el emisor cifra el mensaje con la clave pública del receptor. De esta forma se garantiza que el mensaje sólo puede ser descifrado por el receptor, ya que es el único que conoce la clave privada necesaria. La principal ventaja es que la clave privada nunca viaja ni se comunica, por lo que nadie puede interceptarla.

Veamos un ejemplo: El emisor envía el mensaje cifrado con la clave pública del receptor. El receptor descifra el mensaje con su clave privada

De la misma forma se produce el intercambio de mensajes entre receptor y emisor. El receptor cifra el mensaje con la clave pública del emisor; y el emisor lo descifra con su clave privada.

Como hemos visto, el sistema de criptografía asimétrica ofrece ventajas sobre el de criptografía simétrica. Por este motivo la criptografía asimétrica es la base sobre la que funcionan la autentificación y la firma digital. Pese a ello, este sistema también ofrece alguna desventaja como el tamaño de las claves, el tiempo de proceso o la necesidad de una tercera persona en el proceso, la autoridad certificadora.

En estos sistemas, toda la seguridad recae sobre la complejidad de las claves, por lo que el tamaño de las claves es muy importante. En la actualidad se consideran seguras claves con un mínimo de 1024 bits.

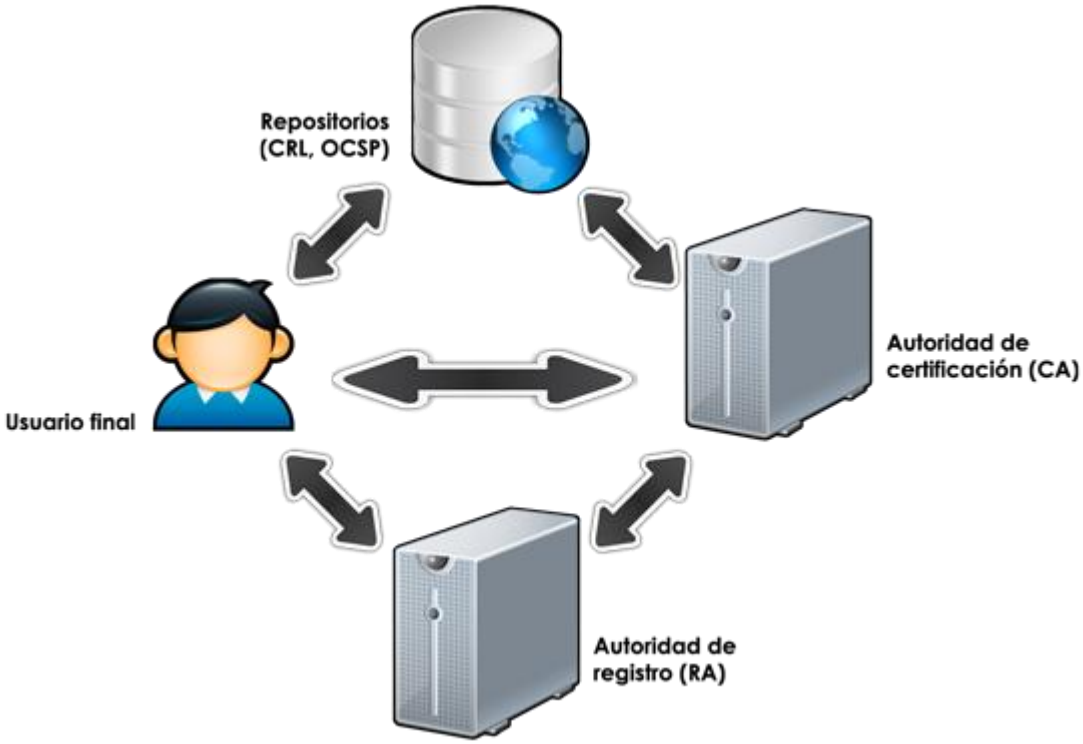
3.6.3. PKI. Infraestructura de clave pública

Una PKI engloba todos los componentes hardware, software, usuarios, políticas y procedimientos que son necesarios para la creación y gestión de los certificados digitales basados en la criptografía asimétrica. Su objetivo es la gestión de las claves y certificados que pueden ser utilizados para propósitos de autentificación, integridad y no repudio.

Un certificado digital es un documento mediante el cual, un tercero confiable (autoridad de certificación o CA) garantiza la vinculación del sujeto con su clave pública.

Los componentes más habituales de una PKI son:

- La autoridad de certificación: (CA) Es la encargada de emitir y revocar certificados. Es la entidad de confianza que da legitimidad a la relación entre la calve pública y el usuario o servicio.
- La autoridad de registro: (RA) Es la encargada de verificar el enlace entre la clave pública y la identidad de sus titulares.
- Los repositorios: Estructuras que se encargan de almacenar la información relativa a la PKI. Los dos más importantes son: el repositorio de certificados y la lista de revocación de certificados, en la que se incluyen todos los certificados que no son validos antes de la fecha establecida.
- Autoridad de validación: (VA) Encargada de comprobar el estado de revocación del certificado
- Autoridad de sellado de tiempo: (TSA) Encargada de sellar con fecha los documentos, para probar que han existido y no han sido alterados desde un instante especifico.
- Los usuarios y entidades finales: Los dueños de las claves privada y pública y su certificado asociado.



El estándar más utilizado para la emisión de certificados es el x509 contiene la siguiente información:

- Número de serie del certificado
- Nombre, dirección y domicilio del suscriptor
- Identificación del suscriptor nombrado en el certificado
- Nombre, dirección y lugar donde realiza su actividad la autoridad de certificación
- Fecha de emisión y expiración del certificado
- Clave pública del usuario
- Metodología para verificar la firma del suscriptor impuesta en el mensaje de datos
- Firma realizada con la autoridad certificadora

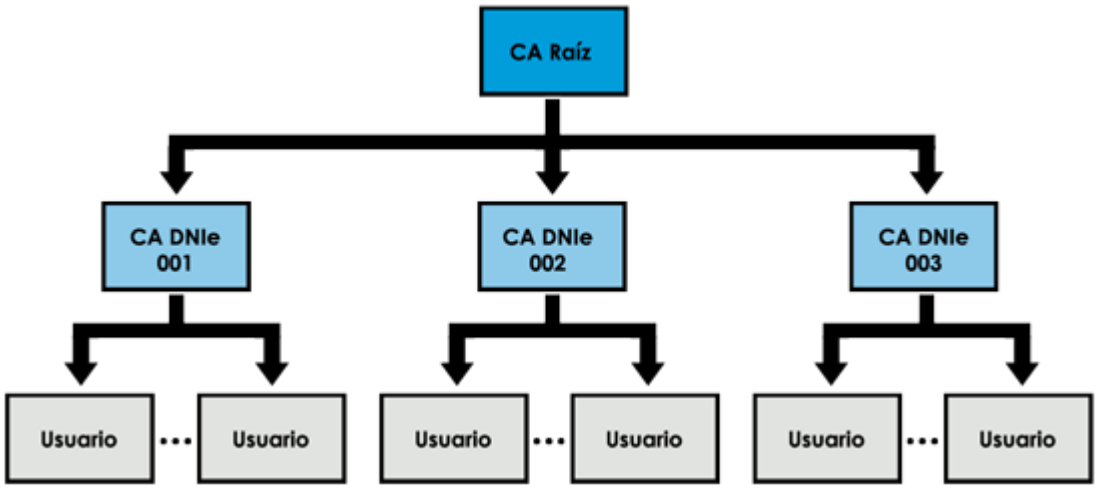
Es responsabilidad de la CA mantener y publicar lista de certificados revocados debidamente. Esto puede ser poco eficiente si la CA es de gran tamaño, por lo que se ha creado el protocolo OCSP, que permite tener lista actualizadas constantemente, pero por contra requiere conexión a internet.

Mientras las CA sean pocas, deberán intercambiar certificados entre ellas, ya que si la CA es grande se producirá una gran carga de procesamiento además de mucho trabajo de administración.

Otra idea importante es la estructura jerárquica. Si cada empleado de una empresa solicitase un certificado a una CA raíz supondría un gasto fuerte para la empresa. Una solución fácil sería que fuese la empres la CA, es decir, que la empresa generase sus propios certificados para sus trabajadores.

3.7. La PKI del DNIe

La arquitectura de la PKI del DNI electrónico sigue la siguiente estructura:



- Un primer nivel en el que se ubica la CA raíz, que representa el punto de confianza de todo el sistema. Permite que todas las personas reconozcan la eficacia de DNIe para acreditar la identidad.
- Un segundo nivel en el que intervienen las CA subordinadas de la CA raíz, que emitirán los certificados de autentificación y firma. Todo DNIe tiene en el chip de la tarjeta, el certificado intermedio de la autoridad de segundo nivel con el que ha sido emitido.

En el proceso de expedición, gestión y utilización del DNIe intervienen varios organismos además de los ciudadanos:

- La dirección general de la policía y guardia civil: Como órgano competente de la expedición y gestión de DNIe
- La Autoridad de aprobación de políticas, encargada de la elaboración, propuesta de aprobación y modificaciones del Proyecto de declaración de políticas de certificación.
- Las autoridades de certificación:
    - CA raíz: Certifica a las CA subordinadas, otorgándoles la posibilidad de firmar (delega) los certificados finales del DNIe

    - CA subordinadas: Su función es la emisión de los certificados del DNIe. Consta de las siguientes:
        - Autoridad de certificación subordinada 001
        - Autoridad de certificación subordinada 002
        - Autoridad de certificación subordinada 003
- La autoridad de registro: (RA) Son todas las oficinas de expedición del documento nacional de identidad. Asisten a la autoridad de certificación en los trámites relacionados con los ciudadanos para su identificación, registro y autentificación. Así se garantiza la asignación de las claves al solicitante.
- Autoridades de validación: (VA) Su función es la comprobación del estado de los certificados. Mediante el protocolo OCSP se determina el estado actual de un certificado de forma rápida y segura.
    - El ministerio de política territorial presta los servicios de validación a las administraciones públicas.
    - La fábrica nacional de moneda y del timbre hace lo mismo para el resto de ciudadanos y empresas.
- Ciudadanos: Los que poseen los certificados.
- Terceros aceptantes: Personas o entidades diferentes del titular que deciden aceptar y confiar en un certificado del DNIe.

3.8. Certificados de autentificación y firma

Como hemos visto, en nuestro chip del DNIe, aparte del certificado de componente, que permite el establecimiento de una canal seguro entre la tarjeta y los controladores; y el certificado de la CA intermedia que ha emitido el DNIe, disponemos de dos certificados x509: el de autentificación y el de firma, ambos con sus correspondientes claves privadas. Estos certificados tienen un periodo de validez de 30 meses. La pérdida o renovación del DNIe lleva aparejada la creación de unas nuevas claves y certificados.

El certificado de autentificación está destinado a validar la identidad del ciudadano a través de medios telemáticos. La autentificación conlleva el uso de la clave privada asociada al certificado de autentificación, de forma que se pueda garantizar que el ciudadano es quien dice ser. El uso de este certificado no está habilitado en operaciones que requieran no repudio de origen (servicio de seguridad que provee al receptor los datos de una prueba del origen de los mismos, que puede usarse ante intentos del emisor de negar su remisión) por lo que los prestadores de servicio no tendrán garantía del compromiso del titular del DNIe. Así pues su uso principal será generar mensajes de autentificación y de acceso seguro a sistemas informáticos. La finalidad del certificado de autentificación es la de garantizar una autenticidad de origen al ciudadano, para que este pueda acreditar su identidad, demostrando la posesión del certificado y el acceso a su clave asociada.

El certificado de firma digital tiene como propósito permitir al ciudadano firmar trámites o documentos, permitiendo la sustitución de la firma manuscrita por la electrónica. Este certificado está incluido en un dispositivo seguro de creación de firma electrónica. Por este motivo, permite la generación de firmas electrónicas reconocidas; es decir, la firma electrónica avanzada que se basa en un certificado reconocido y que ha sido generada utilizando un dispositivo seguro, por lo que se equipara a la firma manuscrita a efectos legales. El uso del certificado de firma electrónica nos proporciona además del no repudio de origen, la integridad, permitiendo comprobar que el documento no ha sido modificado por ningún agente externo desde el momento de su firma.

Por lo anteriormente descrito, el certificado de firma digital no deberá ser usado para crear mensajes de autentificación, ni de acceso seguro a sistemas informáticos. En

definitiva, el uso conjunto de ambos certificados nos proporciona autenticidad de origen, no repudio de origen, e integridad.

Como hemos visto, los certificados de DNIe pueden emplearse para autentificación o firma electrónica de documentos. En ningún caso se contempla el uso de estos certificados y claves para cifrar ningún tipo de información. Además tampoco pueden emplearse como autoridad de certificación.

Los servicios de autentificación del DNIe no han sido diseñados ni autorizados para sistemas de alto riesgo o a prueba de fallos, como hospitales, control de tráfico o cualquier otro sistema en el que un fallo pudiera conllevar la muerte, lesiones personales o daños graves al medio ambiente.

Como hemos visto, el DNIe es un dispositivo seguro de creación de firmas, lo que garantiza que las claves no pueden ser exportadas ni usadas desde otro dispositivo. Las dos claves son generadas bajo la presencia del ciudadano y tienen un tamaño de 2048 bits.

3.9. Comunicación cifrada del DNIe

El DNIe se considera un dispositivo seguro de creación de firmas, ya que cumple el estándar CWA-14890. Este estándar define cómo se realiza la comunicación entre un dispositivo seguro de creación de firmas y una aplicación. Los dos elementos que participan en la comunicación son:

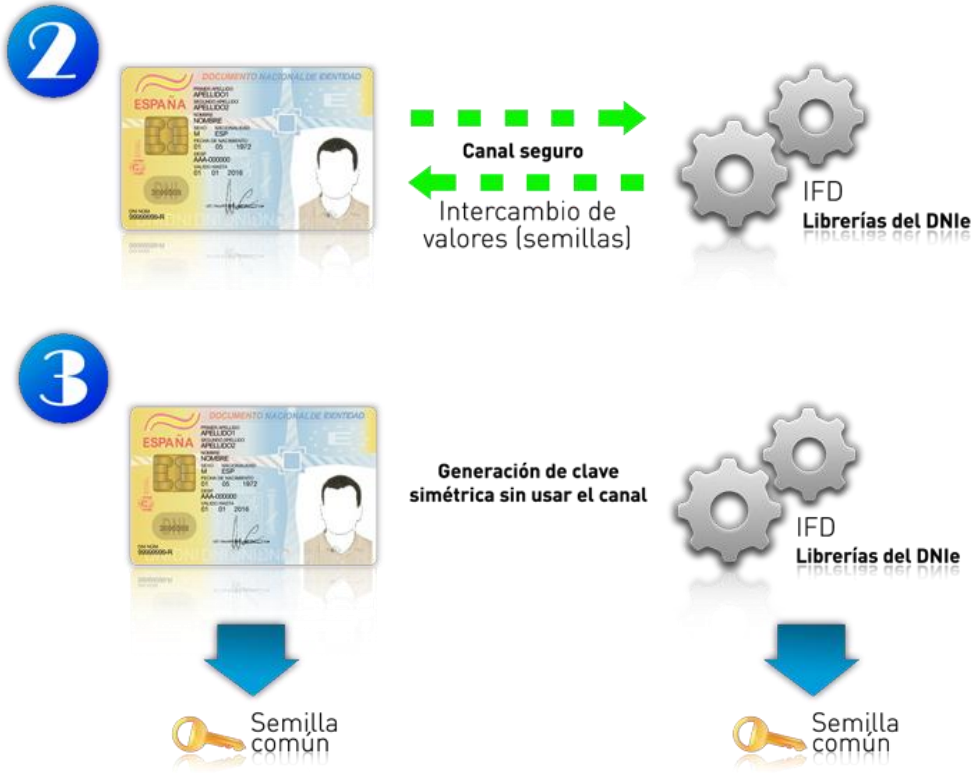
- El terminal o IFD: Las librerías del DNIe.
- El dispositivo o ICC: La propia tarjeta del DNIe.

Para establecer una comunicación, previamente se deben autentificar los dos elementos entre sí. La tarjeta se autentifica con el terminal con su certificado y el terminal se autentifica ante la tarjea con el suyo. Durante esta comunicación, se produce el intercambio de ciertos valores que permiten generar una clave simétrica sin que pase por el canal. Tras generar esta clave, todos los datos que se envían por el canal van cifrados con esta clave. Cualquier comando no cifrado enviado a la tarjeta, o cualquier otra anomalía concluirá con la destrucción del canal seguro y será necesario volver a crearlo de nuevo.

La normativa CWA-14890-1:2004 define cómo establecer este canal seguro y su posterior comunicación cifrada. Con un pequeño esquema explicativo será suficiente para entender este proceso:

En primer lugar se realiza la verificación mutua de certificados. La tarjeta se autentifica ante el terminal (autentificación interna) y del mismo modo el terminal se autentifica ante la tarjeta (autentificación externa).

A continuación se procede al intercambio de unos valores llamados semillas, a través del canal seguro que se está estableciendo. Cada parte genera su clave de sesión.



Una vez establecido el canal, es posible la comunicación cifrada entre tarjeta y terminal y viceversa. El canal ya es seguro, y los datos se mandan cifrados.

Tras una explicación sobre el DNIe, es el momento de centrarse en sus dos grandes funcionalidades, la autentificación y la firma digital. Veremos más en profundidad, en qué consisten y cómo funcionan en el DNIe. Una vez visto esto, tendremos unos conocimientos de base suficientes como para empezar a desarrollar nuestras aplicaciones sobre el DNIe.

3.10. Sistemas actuales de autentificación y firma

Se puede definir la autentificación por el proceso mediante el cual se verifica la identidad de un usuario. Existen tres métodos de autentificación:

- Basados en algo conocido: Contraseñas, frases etc.
- Basados en algo poseído: Tarjetas de identidad, tarjetas inteligentes, dispositivos USB…
- Basados características físicas del usuario: Verificación de voz, huellas dactilares etc.

Dentro de estos grupos, los sistemas más utilizados son los siguientes:

- Autentificación basada en un factor: Un elemento es conocido por el usuario (por ejemplo la contraseña necesaria para entrar en una web)
- Autentificación basada en dos factores: Combina dos métodos de autentificación distintos: hay un elemento que el usuario sabe y otro elemento que el usuario posee. Por ejemplo, el usuario posee el DNIe y además sabe el código Pin de acceso.
- Autentificación biométrica: Consiste en verificar la identidad de un sujeto, basándose en elementos morfológicos que sólo se dan en ese sujeto. Un claro ejemplo es una huella dactilar.
- Autentificación basada en tokens: Son dispositivos para autentificar usuarios y de portabilidad de certificados. Se conectan al ordenador mediante USB, por lo que son compatibles con la gran mayoría de ordenadores y sistemas operativos.
- Además existen otros tipos de autentificación como SSO o LPAD.

La autentificación requiere algoritmos criptográficos y protocolos para estos algoritmos. En general los algoritmos criptográficos se pueden clasificar en tres grandes familias:

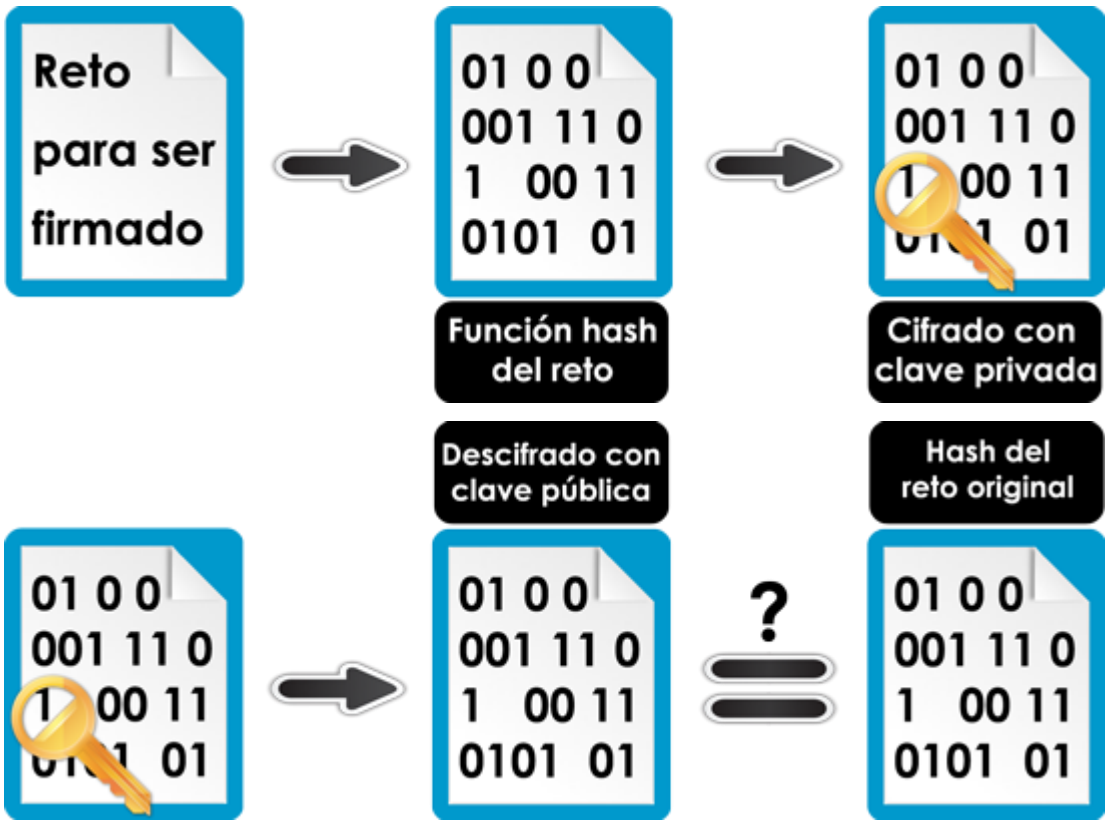
- Criptografía de clave secreta o criptografía simétrica: DES, TDES, RC2,RC4,RC5, AES …
- Criptografía de clave pública o criptografía asimétrica: RSA, DSA entre otros.

- Algoritmos hash o de resumen: SHA, MD5 etc.

Todos estos algoritmos siguen diferentes protocolos como PGP, SSL, TLS, SSH…

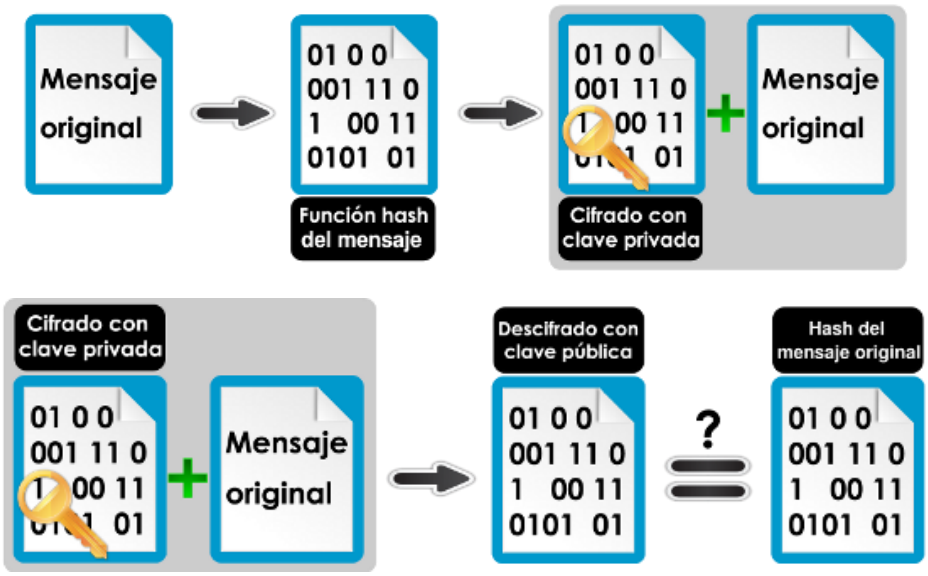
La autentificación con certificado se basa en el protocolo reto respuesta, donde una parte presenta un reto y la otra parte contesta al reto con una respuesta que necesita ser autentica. Más concretamente:

- Participan dos partes: El usuario que necesita ser autentificado y la parte encargada de realizar la autentificación.
- Cada parte implicada tiene dos claves. Una clave pública conocida por la otra parte y una clave privada que sólo conoce el propietario.
- La parte encargada de la autentificación envía un reto a la otra parte.
- El usuario calcula un resumen hash de los datos y lo cifra con su clave privada. Después le envía estos datos a la parte encargada de la autentificación.
- Por último, la parte encargada de la autentificación descifra los datos con la clave pública del usuario y verifica que el resumen de los datos que ha enviado el usuario coincide con el resumen de los datos que había calculado.



La firma electrónica es un conjunto de datos asociados a un mensaje, y que permiten verificar su integridad y la identidad del firmante. Está basada en la criptografía asimétrica o de clave pública y funciona así:

- El firmante dispone de un certificado con sus claves asociadas, una pública y otra privada. La clave del certificado es también conocida por el receptor.
- El firmante obtiene un resumen hash de los datos a firmar y los cifra con su clave privada. El resultado es la firma electrónica que se añade al mensaje original.
- Una vez enviados los datos, el receptor descifra la firma adjuntada usando la clave pública del emisor, para obtener el resumen realizado por el firmante. Por otro lado realiza el resumen hash de los datos originales. Se contrastan los dos resúmenes, el recibido y el generado. En caso de ser idénticos, la firma es correcta, garantizándose la identidad de los datos así como la identidad del firmante. En caso contrario, los datos no tienen ninguna validez ya que demuestran que los datos han sido vulnerados.
-



Hay diferente tipos de firma electrónica:

- Firma básica: Contiene un conjunto de datos recogidos de forma electrónica, que formalmente identifican al autor y se incorporan al propio documento.
- Firma avanzada: Es el tipo de firma que permite identificar al firmante y detectar cualquier cambio en os datos firmados. Está vinculada al firmante de manera única

y a los datos a que se refiere. Debe crearse a través de medios que el firmante pueda mantener bajo su exclusivo control.

- Firma reconocida: Tiene las mismas características que la firma avanzada, pero está basada en un certificado reconocido y ha sido generada bajo un dispositivo seguro de creación de firmas. Tiene el mismo valor que una firma manuscrita.
- Firma fechada: Firma electrónica a la que se ha añadido un sello de tiempo. El sellado de tiempo permite demostrar que una serie de datos han existido y no han sido alterados desde un instante específico en el tiempo.
- Firma validada: Firma electrónica fechada, a la que se ha añadido información sobre la validez del certificado procedente de una consulta OCSP a la autoridad de validación.
- Firma longeva: Firma electrónica validada y dotada de validez a lo largo del tiempo. Esto se consigue al ir refirmando y actualizando los sellos de tiempo de forma regular. De esta manera unos datos que fueron firmados en su día con un algoritmo válido pero que ya no lo es por los avances tecnológicos, no pierden valor al ser refirmados de forma segura.

Como hemos visto el DNIe produce una firma avanzada y reconocida. Estos son los tipos de formatos básicos para este tipo de firmas:

- PKCS7/CMS: Este formato se utiliza para firmar, resumir, autentificar o cifrar el contenido de un mensaje.
- Firma XML: Este formato es usado frecuentemente en aplicaciones on-line. Es parecido a CMS pero la codificación original de firmas y certificados se realiza en base 64.
- S/MIME: Es un estándar para criptografía de clave pública y firmado de correos electrónicos encapsulado en MIME. Ofrece autentificación, integridad del mensaje y no repudio.
- XAdES: Es un conjunto de extensiones a las recomendaciones XML-DSig, haciéndolas adecuadas para la firma electrónica avanzada. XAdES especifica perfiles precisos de XML-DSig para ser usados como firma reconocida. Este tipo de firma es el que se usa para la creación de facturas electrónicas.

3.11. Algoritmo RSA

En la actualidad, RSA es el primer y más utilizado algoritmo de firma y cifrado. Su seguridad radica en el problema de factorización de números enteros. Los mensajes enviados se representan mediante números, y el funcionamiento se basa en el producto, conocido, de dos números grandes elegidos al azar y mantenidos en secreto. Actualmente estos primos son del orden de 10^200, y se prevé que su tamaño crezca con el aumento de la capacidad de cálculo de los ordenadores.

El algoritmo RSA se divide en tres pasos:

- Generación de claves:
    - Cada usuario elige dos número primos aleatorios distintos p y q, pero con longitud en bits parecida.
    - Se calcula el módulo n=p*q
    - Se calcula la función de Euler (?) del modulo ?(n)=(p-1)*(q-1)
    - Se escoge un entero positivo e (exponente de clave pública) menor que ?(n), que sea comprimo ?(n). Un exponente muy pequeño puede suponer un riesgo para la seguridad.
    - Se determina un d (exponente de la clave privada) que satisfaga la concurrencia de: ? 1(mod ? (n)).
    - Entonces, (n,e) será la clave pública y (n,d) la clave privada mantenida en secreto.
- Cifrado:
    - El emisor A comunica su clave pública al receptor B y guarda la clave privada en secreto. Ahora B desea enviar un mensajes secreto M a A.
    - B convierte M en un número entero m menor que n mediante un protocolo reversible acordado de antemano.
    - Luego B, calcula el texto cifrado c mediante la operación m^e ? (mod n) y transmite c a A.
- Descifrado
    - A puede recuperar m a partir de c, usando su exponente d de la clave privada: c^d ? m(mod n)

- Ahora A puede recuperar el mensaje original M invirtiendo el esquema de relleno. RSA debe ser combinado con alguna versión del esquema de relleno, ya que si no el valor de M puede llevar a textos cifrados inseguros.

RSA es mucho más lento que DES y que otros sistemas simétricos. En la práctica, el receptor normalmente cifra mensajes con algoritmos simétricos, cifra la clave simétrica con RSA, y transmite a ambos la clave simétrica RSA cifrada y el mensaje simétricamente cifrado al emisor. Esto plantea problemas adicionales de seguridad, por lo que es muy importante usar un generador aleatorio fuerte ya que si no, el atacante podría puentear la clave simétrica de RSA mediante la adivinación de la clave simétrica.

3.12. Datos personales en los certificados

Ya hemos visto en qué consiste una autentificación o una firma digital, y también que nuestro DNIe posee unos certificados especialmente diseñados para realizar estas operaciones. Ahora veamos como comprobar la identidad de una persona que presente un certificado y demuestre tener acceso a su clave privada correspondiente.

Leyendo la información del certificado podemos extraer información acerca de la persona. El campo más importante es el subject, que contiene la siguiente información:

- CN: Con el siguiente formato: Primer Apellido – Segundo Apellido – Nombre (Autentificación) o (Firma)
- Serial Number: Que contiene el NIF del ciudadano.

3.13. Desarrollar con el DNIe:

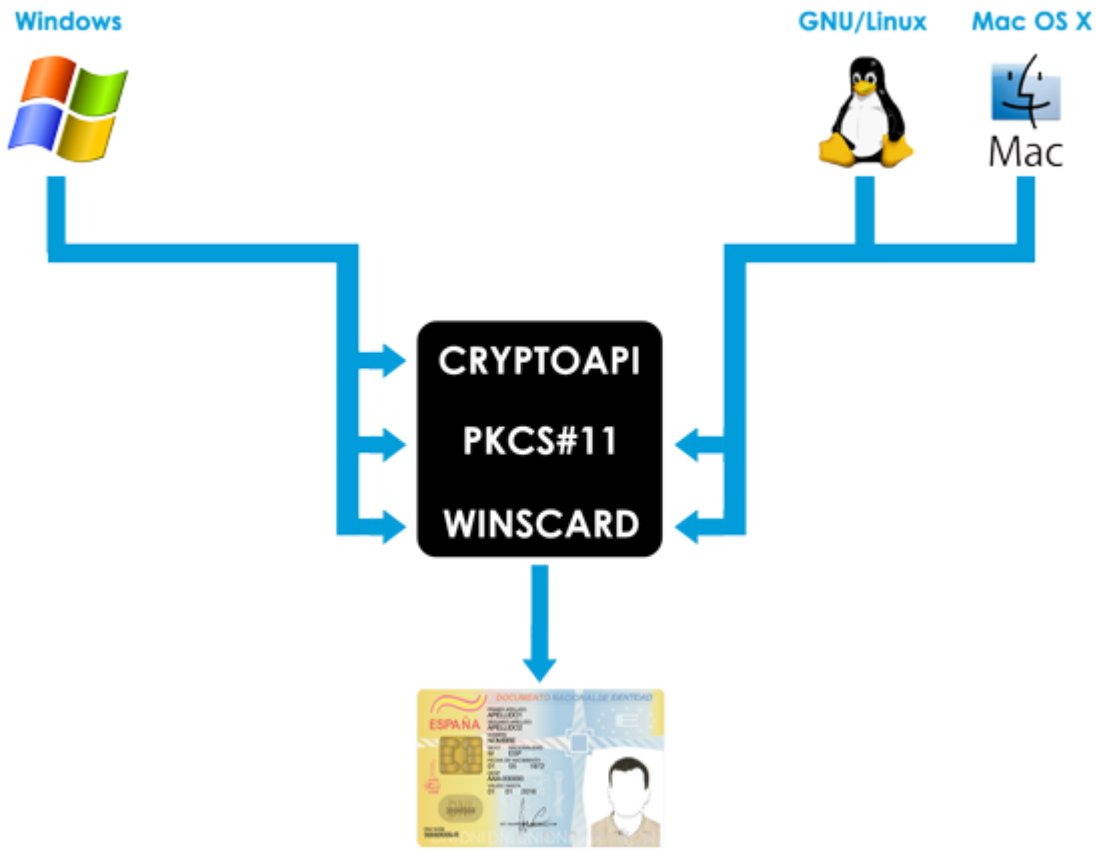
Ahora que ya tenemos claro qué es un DNIe, y que se puede hacer con él, es el momento de saber algo más sobre cómo se desarrollan aplicaciones para el DNIe. Lo primero que debemos plantearnos es a qué nivel vamos a trabajar con el DNIe. Si deseamos trabajar a nivel de certificados es necesario el uso de los controladores oficiales del DNIe además del conocimiento del código Pin. Si por el contrario sólo se desean extraer algunos datos públicos del DNIe como el nombre apellidos o número de DNI, se podrá acceder a estos datos mediante el envío de algunos comandos y sin la necesidad de conocer el código Pin.

La mejor opción es trabajar a nivel de certificados, ya que a día de hoy se consideran seguros los algoritmos empleados para la emisión de certificados. Por el contrario, se podría construir una tarjeta falsa, con la misma estructura que un DNIe, que emitiera unos datos públicos falsos. Si a esto unimos la mayor potencia que otorga trabajar a nivel de certificados, la elección no puede ser otra que trabajar a nivel de certificados.

Para la realización de aplicaciones que no requieran acceso a los certificados se utiliza la librería WinSCard, que accede a la información directamente enviando comando APDU a la tarjeta.

Los desarrollos que vamos a realizar serán de nivel criptográfico. Estos desarrollos pueden realizarse mediante dos liberarías:

- CryptoApi: Sólo disponible en sistemas Microsoft Windows. Utilizan sistemas nativos de Microsoft como Internet Explorer o Outlook.
- API PKCS#11: De carácter multiplataforma y compatible con otras aplicaciones y métodos de autentificación y firma.



En este caso, vamos a utilizar la API PKCS#11 ya que todos los sistemas operativos soportados por el DNIe poseen una implementación del módulo PKCS#11. De esta forma, desarrollando un solo código, tras compilarlo en la plataforma adecuada, debería funcionar sin la necesidad de cambiar ni una línea de código, al menos en lo referente al acceso al módulo. La implementación de estos módulos las podemos encontrar en las siguientes rutas:

- Microsoft Windows: C:WindowsSystem32UsrPkcs11.dll
- GNU/LINUX: /usr/lib/opensc-pkcs11.so
- MAC OS X: C:WindowsSystem32UsrPkcs11.dll

Otro tipo de desarrollos frecuentes son las aplicaciones web. Cuando se desea establecer un sistema de control de acceso a una web mediante certificados, los más seguro y habitual es utilizar un sistema de autentificación mutua SSL. Tanto el servidor web como el navegador se autentifican el uno frente al otro con certificados electrónicos garantizando así la identidad de ambas partes. El protocolo SSL es muy utilizado hoy día, por lo que

todos los servidores web se pueden configurar sin problemas para realizar este tipo de operaciones. Sea cual sea el servidor elegido, el servidor se encarga de realizar automáticamente la validación local del certificado, pero suele ser necesario que el desarrollador de la web realice la comprobación del estado del certificado mediante el protocolo OCSP, ya que esta funcionalidad no está integrada por defecto en los servidores. Como servidor web, usaremos Apache, ya que es el más extendido, y además es bastante eficiente para conexiones https, que son las que queremos hacer.

También hay que tener en cuenta que si la aplicación web requiere acceso a los recursos del sistema del cliente, será necesario el desarrollo de aplicaciones web que se ejecuten desde el navegador. Una buena opción son los Applets de java, con soporte para la gran mayoría de navegadores del mercado, además de ser multiplataforma. Hay que tener en cuenta que para que un Applet tenga acceso a recursos de la máquina, como el lector de tarjetas, será necesario que esté firmado por un certificado emitido a tal efecto. Otra opción es utilizar las funciones que trae el navegador para realizar estas funciones. Elegiremos esta segunda opción, que es más sencilla de implementar y no requiere de firmas o certificados. Además crear y firmar certificados será algo necesario para la autentificación del servidor, por lo que ese tema también estará tratado.

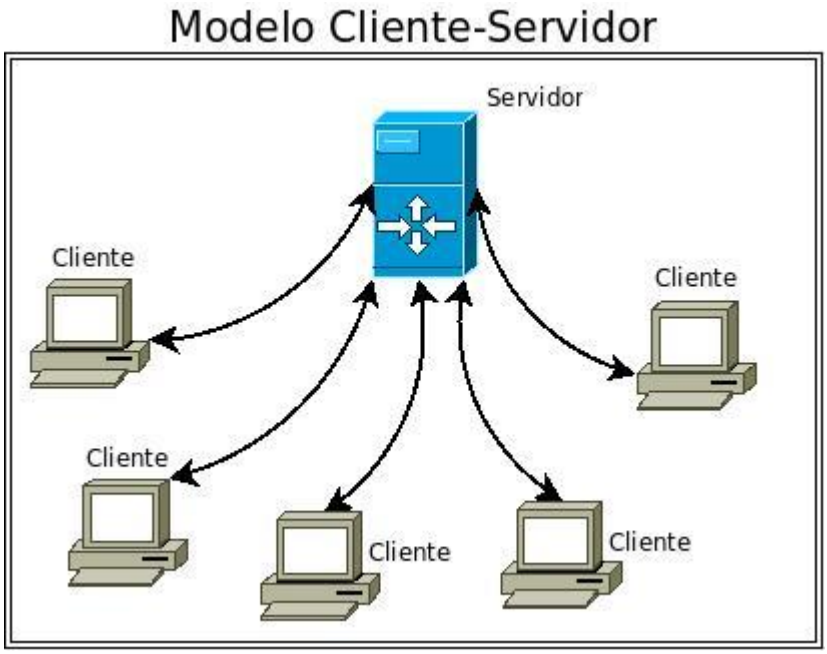
De todas formas, veremos todos estos temas más en profundidad a la hora de desarrollar nuestras aplicaciones.

Ahora mismo, ya poseemos los conocimientos necesarios para desarrollar correctamente nuestras aplicaciones, por lo que ya podemos comenzar a desarrollar.

3.14. Aplicación 1: Chat

El objetivo es desarrollar un chat, en el que los usuarios tengan que autentificarse mediante el uso del DNIe. Además todos los mensajes viajarán cifrados, para que ninguna persona ajena al chat pueda leer las conversaciones.

Para ello utilizaremos un servidor central, que se encargará de autentificar a los clientes y de reenviar todos los mensajes que lleguen. De esta forma evitamos que un usuario deba autentificarse ante cada persona que se conecte al chat.



Como hemos visto, la autentificación en páginas web se realiza, generalmente, mediante el protocolo SSL. Para nuestro chat utilizaremos este protocolo, que nos permitirá autentificar tanto al usuario como al servidor y cifrar todos los mensajes. La autentificación del cliente se realizará mediante el DNIe, pero tenemos que tener en cuenta que el servidor también se debe autentificar ante el cliente para evitar ataques de Man in the middle.

Estos ataques consisten en hacer creer al cliente que está conectado con el servidor que desea, cuando realmente esta intercambiando mensajes con un atacante. Para evitarlos, la clave pública se entrega junto con un certificado firmado por una autoridad certificadora, que conocen ambas partes, y que identifica al poseedor de la clave.

Así pues, aparte de las claves y certificados del DNIe, necesitaremos otro par de claves con su correspondiente certificado. Este será nuestro primer paso.

3.14.1. Generación del claves y certificados

Para crear nuestras claves y certificados usaremos el programa OpenSSL. Tenemos dos opciones: Crear un certificado auto-firmado, es decir, ser nosotros mismos nuestra autoridad certificadora; o bien solicitar que una autoridad certificadora reconocida en internet firme nuestro certificado. Esta segunda opción es mejor, ya que todo el mundo en internet podrá reconocer nuestro certificado como válido. Por este motivo, es más recomendable utilizar esta opción en servicios en internet. Pero para mostrar un ejemplo de funcionamiento, no es necesario, por lo que nosotros mismos seremos nuestra propia autoridad certificadora.

Lo primero que haremos será crear un par de archivos que formarán la base de datos de los certificados auto-firmados:

- Echo '01' > serial
- > index.txt

El primer archivo 'serial' contiene el número de serie de nuestros certificados, el primero será el 01, y tras crear un certificado se incrementará en una unidad. El otro archivo 'index.txt' será la base de datos propiamente, en base al número de serie.

Para crear certificados, openssl ofrece un mecanismo de entrada de datos a través de teclado, pero esto no es muy recomendable si se van a crear varios certificados, ya que tendrás que repetir los datos una y otra vez. Para evitar esto, podemos editar un archivo 'openssl.cnf' que contenga la configuración deseada, simplificándonos la creación de certificados. Configurar este archivo es muy intuitivo, ya que sigue la estructura de variable=valor. La variable más importantes es distinguised_name(DN) o nombre distinguido, que hace referencia a una sección que tiene los datos básicos de la autoridad certificadora o CA. Además del nombre, agregaremos otros datos como email, ciudad, estado o país.

Todo está casi listo para crear el certificado raíz, el que nos convertirá en una autoridad certificadora, así que lo primero que nos pedirá cuando introduzcamos el comando es el "pass phrase". Esta contraseña es de vital importancia, ya que es con la que

validaremos nuestra autoridad certificadora para poder crear los certificados. Debe ser preferentemente muy compleja, con mayúsculas, minúsculas, espacios, números y símbolos. Aunque sea difícil de recordar, es necesario que sea una contraseña fuerte, ya que los algoritmos de cifrado son sumamente difíciles de romper mediante fuerza bruta, por lo que la verdadera debilidad es el uso de contraseñas débiles. Ahora tenemos todo listo, incluyendo una buena contraseña.

- openssl req –new –x509 –extensins v3_ca –keyout privado/cakey.pem –out cacert.pem –days 3650 –config ./openssl.cnf

Las opciones incluidas son: req –new –x509 para crear un certificado nuevo auto-firmado. –extensions v3_ca para crear un certificad raíz CA. –keyout privada, nombre y donde guardará la llave. –out indica el nombre del certificado raíz CA. –days 3650 tiempo de validez del certificado y con –config indicamos el archivo de configuración a utilizar. Esto da como resultado dos archivos:

- Un certificado raíz CA (cacert.pem)
- Una llave privada privada/cakey.pem)

El archivo cacert.pem será el que se podrá mandar a nuestros clientes o usuarios de nuestro sistema. Si abrimos este fichero con un editor de textos, no veremos nada más que un conjunto de letras, números y símbolos, pero eso no quiere decir que no esté la información que nosotros hemos puesto. Podemos consultar estos datos a través del subcomando x509 de openssl. Por el contrario, el contenido de una llave privada jamás debe publicarse, mostrarse o mandase a nadie.

En este punto, ya tenemos un certificado raíz que nos valida como CA, sin más autoridad que nosotros mismos por supuesto. Lo siguiente que debemos hacer crear una solicitud de firmado de certificado o CSR, para ello, crearemos una llave privada y una solicitud de certificado y firmaremos la solicitud para generar un certificad auto-firmado. El procedimiento será igual al anterior. Sólo que en la solicitud de firmado no es necesario especificar una contraseña.

- openssl req –new –nodes –out servidor-cert.pem –config ./openssl.cnf

Que generará dos archivos: servidor-cert.pem será la solicitud de firmado de certificado; y key.pem que será la clave privada de este certificado. Tras verificar el contenido de nuestra solicitud de certificado, será el momento de firmar la solicitud. Para ello deberemos indicar la contraseña que autentifique que somos la CA, lo que nos da derecho a firmar certificados.

- openssl ca –out certificado-servidor.pem –config ./openssl.cnf –days 3650 –infiles servidor-cert.pem

A destacar, que ahora usamos la opción ca, para indicar que firmaremos un certificado como autoridad certificadora. La salida –out será el archivo del certificado, usando las opciones del archivo de configuración y la solicitud de firmado que se especificó en –infiles. Si ahora comprobamos el contenido del fichero 'serial.txt' que creamos al principio veremos que se ha actualizado a 02. El archivo 'index.txt' además del número de serie 01 indica los campos del DN.

Pero por supuesto, lo que más nos interesa son los archivos: 'certificado-servidor.pem', correspondiente al certificado auto-firmado. Y el archivo 'key.pem' que será su llave privada. Con esto ya tenemos todas las calves y certificados necesarios para nuestro servidor, pero aunque los certificados y claves del cliente los extraeremos del DNIe, es más que conveniente usar primero unos archivos normales como clave y certificado para los clientes, con el fin de aislar más los posibles fallos. Por este motivo, creamos otro certificado auto-firmado con su clave correspondiente, esta vez para un cliente.

3.14.2. s_server y s_client

Antes de lanzarnos a programar sin pensar, miramos un poco más en detalle todas las posibilidades que nos ofrece el programa openssl, que no son pocas. De entre ellas, nos llaman la atención dos, s_client y s_server. Básicamente estas funcionalidades sirven para probar la respuesta de servidores ante distintas peticiones de clientes. Dado que han fin de cuentas, lo que queremos hacer es un servidor central que autentifique a los clientes, no está demás probar un poco estos comandos. Leyendo un poco el manual de usuario, podemos configurar el servidor tal y como queremos:

- openssl s_server –cert server.crt –key server.key –Verify 1

Es decir, un servidor que se autentifique ante los clientes con el certificado y clave que acabamos de crear y que exija que los clientes también se autentifiquen. Al tratarse de un ejemplo para probar que funciona, el servidor no verificará que el certificado del cliente este firmado por una CA en concreto, aunque también hay opciones para configurarlo. Para lanzar el cliente, la sintaxis es muy parecida:

- openssl s_client –connect 127.0.0.1:4433 –cert certificado-cliente.pem –key clave-cliente.pem

Como vemos es muy parecido al anterior, sólo que esta vez tenemos que indicar a que dirección IP y puerto conectar. El resto es igual, cargamos el certificado y la clave del cliente, pero no verificamos que el certificado del servidor sea válido.

Si ejecutamos ambos comando, vemos que realmente funciona, y que podemos intercambiar mensajes entre servidor y cliente de forma cifrada. Para que la autentificación sea completa, el cliente debería comprobar el certificado del servidor, y el servidor el del cliente. Para ello utilizaríamos la opción –CAfile, indicando el archivo correspondiente al certificado de la autoridad certificadora que hemos creado.

Una vez probado que funciona correctamente, podríamos implementar nuestros programas utilizando la librería de funciones de openssl, y configurarlos para que funcionen de manera análoga. Esto no es demasiado complicado, ya que hay multitud de

ejemplos en internet que realizan esto mismo. Únicamente tendríamos que modificar un poco el servidor para que aceptase conexiones de múltiples clientes, y hacer que lo que escriba cada cliente al servidor, este lo reenvíe a todos los usuarios conectados.

Haciendo esto, ya tendríamos el programa que queríamos, sólo que el cliente utilizaría sus certificados y claves desde a archivos y no desde el DNIe que realmente es lo que queremos. Así pues, lo que debemos plantearnos es cómo cargar los certificados y claves del DNIe en las librerías de openssl.

3.14.3. El proyecto OpenSC:

Desgraciadamente, openssl no tiene por defecto soporte para tarjetas criptográficas. Para poder usar tarjetas criptográficas, como el DNIe, en openssl es necesario el uso de un engine. En este caso el engine PKCS#11 desarrollado por OpenSC.

El proyecto OpenSC provee una serie de librerías y funciones para trabajar con tarjetas criptográficas, facilitando su uso en aplicaciones de seguridad que usen autentificación, encriptación o firma digital.

Dentro del proyecto OpenSC podemos encontrar algunas sub-proyectos muy interesantes:

- Engine PKCS#11: Es un engine para OpenSSL que da soporte para tarjetas criptográficas.
- Libp11: Pequeña librería de funciones en C, que hace más fácil a los desarrolladores utilizar las librerías PKCS#11.
- OpenSC-Java: Librería para usar las tarjetas criptográficas en Java.
- Pam_p11: Facilita el uso de las tarjetas para autentificación local en un ordenador.

Como lo que queremos es dar soporte para tarjetas criptográficas a OpenSSL descargaremos El Engine PKCS#11 de la página del proyecto: www.opensc-project.org

Hay que tener en cuenta, que lo que hemos descargado no es un archivo empaquetado y listo para instalar, sino que hemos descargado el código fuente, por lo que para instalarlo deberemos compilarlo antes. Esto no supone tampoco un gran esfuerzo, ya que basta con abrir una terminal, situarse en la carpeta correspondiente y hacer ‘./configure’, ‘make’ y ‘make install’.

Una vez instalado, si todo ha ido bien, es el momento de probarlo con nuestro DNIe. Para ello volveremos a lanzar los comandos de openssl s_client y s_server. Cómo ya hemos visto, todavía no nos vamos a preocupar de que los certificados estén firmados por una autoridad certificadora o por otra, ya que sólo nos interesa cargar con éxito los

certificados y claves para comprobar que funciona. Por tanto, el funcionamiento del servidor será el mismo:

- openssl s_server –cert server.crt –key server.key –Verify 1

Pero lo que si debemos de cambiar es el cliente. Para aislar los posibles problemas, extraemos el certificado de autentificación del DNIe a un archivo. Esto es muy fácil de hacer, por ejemplo con Firefox. Sólo tenemos que ir a ver los certificados de cliente que tenemos y exportar a archivo el certificado de autentificación. No podremos hacer los mismo con la clave privada, ya que como hemos visto nunca abandona el DNIe. Cada clave privada tiene un identificador que necesitamos conocer para poder referirnos a ella. Esta y otra información pude obtenerse fácilmente mediante el comando:

- pkcs15-tool –dump

Además del id de las claves y certificados, podemos ver sus nombres y el uso para el que están destinadas (autentificación o firma). Una vez sabemos el id de nuestra clave privada podemos lanzar el comando para el cliente. Hay que tener en cuenta, que para poder utilizar el engine, primero hay que cargarlo:

- openssl engine –t dinamyc –pre SO_PATH:/usr/lib/engines/engine_pkcs11.so \
- -pre ID:pkcs11 -pre LIST_ADD:1 -pre LOAD \
- -pre MODULE_PATH:/usr/lib/opensc-pkcs11.so

Tras cargar el engine, obtendremos un mensaje de carga exitosa, y ya podremos lanzar nuestro comando usando este engine.

- Openssl s_client –engine pkcs11 –connect 127.0.0.1:4433 –cert certAUTENTIFICACION –key id_413036303641323532363032364130323031303013230313032343035 –keyform engine

De esta forma conectamos con esa dirección ip y ese puerto usando como certificado el certificado de autentificación del DNIe (que lo hemos guardado en un fichero) y usamos la clave privada almacenada en nuestro DNIe.

Por desgracia, el único mensaje que obtendremos será un error diciendo que nuestra tarjeta no está soportada. El problema radica en que estamos sobrecargando las llamadas a la tarjeta a través del Engine. Para hacerlo debemos utilizar directamente la librería PKCS#11. En otras palabras, este engine todavía no da soporte para la tarjeta del DNIe. El problema es que el código fuente del DNIe no ha sido liberado completamente, por lo que OpenSC no puede dar soporte a esta tarjeta debido a un incumplimiento de licencias. De esta forma, la liberación total del código fuente del DNIe traerá a la larga la integración en el proyecto OpenSC lo que supondrá sin duda una gran ventaja para su integración en diferentes aplicaciones de una manera más fácil.

Así pues, si queremos interactuar con nuestro DNIe tendremos que trabajar directamente con la API de funciones de la librería PKCS#11, lo que supondrá un esfuerzo extra, pero también nos ayudará a entender mejor cómo funciona el DNIe internamente y a comprender cómo desarrollar aplicaciones para nuestro DNIe.

La solución más sencilla es la siguiente: Implementamos un servidor SSL que se autentifique ante los clientes con su clave y su certificado como hasta ahora. Una vez que el cliente verifica que el servidor es realmente quien dice ser se establece un canal seguro entre ambas partes. Tras esto el servidor solicitará al cliente que se autentifique. Para ello el servidor generará un reto que el cliente debe entregar firmado junto con su certificado. Para que la autentificación sea válida, el servidor deberá comprobar que el reto ha sido firmado con la clave privada correspondiente al certificado, comprobar que el certificado este firmado por la autoridad certificadora y además hacer una consulta OCSP para contrastar que el certificado no esta revocado. Si no se verifica uno de estos 3 pasos no se completa la autentificación. En caso de que la autentificación se complete el cliente podrá intercambiar mensajes con todos los usuarios autentificados en el chat.

3.14.4. Desarrollo de la aplicación

Ya hemos visto que para el desarrollo que queremos hacer, debemos utilizar la API PKCS#11. Sin embargo, el proyecto OpenSC incluye unas librerías libp11 para hacer más fácil el desarrollo de aplicaciones en leguaje C. También incluye OpenSC-java para poder desarrollar aplicaciones Java para el DNIe. Desarrollar en Java tiene ventajas, ya que es un lenguaje de más alto nivel, y resulta más cómodo a la hora de programar e incluso permite generar interfaces gráficos más fácilmente. Pero utilizar Java implica utilizar sus clases, que funcionan, sin ser tan necesario entender qué estamos haciendo realmente. Por ejemplo, es muy común que en una sola instrucción en Java realices varias cosas a la vez. Por el contrario, desarrollar en C supone una mayor dificultad a la hora de programar, sin embargo, esto supone un mayor entendimiento final del programa que has desarrollado. Además las librerías de OpenSSL están en C por lo que podemos trabajar directamente con estas, mientras para hacer una conexión SSL en java tendríamos que usar clases especiales exclusivas de Java. Como el objetivo del proyecto es comprender el funcionamiento del DNIe, y las técnicas de criptografía, parece más interesante realizar el desarrollo en C. Esto supone sacrificar un poco el interfaz de usuario, pero ya tendremos tiempo de hacer un interfaz más amigable en la segunda aplicación, ya que diseñaremos una página web.

Una vez que hemos decidido que librerías vamos a usar, debemos instalarlas. Para instalar la librería libp11 deberemos descargarla desde su página oficial: www.opesc-project.org Para la instalación, al igual que antes bastará con hacer ‘./configure’ ‘make’ y ‘make install’. Otras librerías que necesitaremos serán las de desarrollo de OpenSSL. Para instalarlas basta con buscar e instalar el paquete openssl-dev. Ya tenemos todo lo necesario, por lo que es el momento de empezar a desarrollar.

3.14.4.1. Control de versiones:

Subversion es un sistema gestor de versiones. Se usa para mantener el historial de ficheros tales como código fuente, páginas web y documentación. Aunque cuando verdaderamente es necesario el uso de este tipo de programas es cuando estás desarrollando en grupo, también nos ofrece ventajas a la hora de desarrollar un proyecto como el que estamos haciendo. La principal ventaja es que mantienes en un servidor la última versión de tu código, por lo que acceder a ella desde cualquier ordenador es muy simple. Esto facilita que el tutor pueda visionar siempre la última versión del código que esté desarrollando el alumno. Para hacer algo tan básico como esto nos bastará con conocer algunas de las órdenes más importantes del Subversion.

Lo primero que debemos hacer es crear un repositorio. Esto creará una nueva carpeta que contendrá varios ficheros y carpetas que sirven para cambiar la configuración del repositorio creado.

- svnadmin create nombre_del_repositorio

Ahora haremos un envío inicial al repositorio. Una de las formas más sencillas de acceder a nuestro repositorio es el protocolo svn+ssh. Como su propio nombre indica, este protocolo conecta con el repositorio de subversión a través de ssh.

- svn import svn+ssh://usuario@servidor/path ruta_local –m “Importacion Inicial”

De esta forma importaremos lo que tengamos en ruta local al repositorio. Importación inicial es sólo un mensaje que saldrá al consultar el log del repositorio. Ahora ya tenemos nuestro proyecto subido al servidor.

La forma normal de trabajar es realizar una descarga del proyecto, realizar los cambios que deseemos y subir estos cambios al repositorio. Para hacer una descarga inicial del proyecto crearemos una carpeta vacía en la que descargaremos el proyecto.

- mkdir carpeta_proyecto
- svn checkout svn+ssh://usuario@servidor/patth ./carpeta_proyecto

Una vez hecho esto se habrá descargado todo lo que haya en el repositorio y unas caretas ocultas que controlan los cambios en el repositorio. Ahora ya podemos hacer cambios en los archivos copiados y Subversión se encargará de vigilar los cambios. Para controlar que archivos hemos cambiado podemos usar el comando

- svn status

Este comando nos dirá, que archivos hemos cambiado respecto al repositorio. Además de modificar los archivos que ya existían, podemos añadir nuevos archivos, borrarlos o moverlos. Una vez hechos todos los cambios será el momento de actualizar el repositorio, para ello basta con hacer:

- svn comit –m “ejemplo de cambios”

De esta forma ya habremos actualizado el repositorio, y ahora cualquier usuario que descargue el proyecto verá la versión con las modificaciones que hemos hecho.

3.14.4.2. Cliente

Como ya hemos visto, para desarrollar nuestro programa necesitaremos un servidor central y varios clientes. El programa cliente es más sencillo de implementar y comenzaremos por él. Explicaremos su funcionamiento mostrando las líneas de código más interesantes y no el código fuente completo, ya que este se entrega adjunto con el documento.

Tras incluir todas las librerías necesarias y definir las variables, será el momento de recoger los datos que el usuario nos manda al iniciar el programa. Para que el programa sea ejecutado con éxito, el usuario deberá introducir como primer argumento la ip del servidor, y como segundo el puerto en el que está escuchando.

```
if(argc!= 3){
        printf("Uso : %s <ip> <puerto>\n", argv[0]);
        exit(1);
}

short int s_port = atoi(argv[2]);
const char *s_ipaddr =argv[1];
```

Si el usuario no lanza el programa con esa sintaxis, dará un error y saldremos del programa. Si lo ha lanzado correctamente guardamos la dirección ip y el puerto para posteriormente conectar a ellos.

A continuación vamos a cargar y configurar las librerías de openSSL:

```
SSL_library_init();
  /* Cargar el strings de error de SSL */
SSL_load_error_strings();

 /* Elegimos la version 3 de SSL */
meth = SSLv3_method();

 /* Creamos un contexto*/
ctx = SSL_CTX_new(meth);

RETURN_NULL(ctx);
```

Tras cargar los algoritmos de SSL y un string para imprimir los posibles fallos creamos un contexto. Es importante elegir la versión de SSL en el que se va a realizar la conexión. Elegimos la versión 3, que actualmente se considera segura. Tras elegir estas opciones crearemos el contexto. Necesitamos el contexto para poder crear la estructura SSL que asignaremos al socket, pero antes de ello configuraremos la estructura.

```
/* Cargamos el certificado de la CA en la estructura */
/* Esto nos permite verificar el certificado del servidor     */

if(!SSL_CTX_load_verify_locations(ctx, RSA_CLIENT_CA_CERT, NULL)){
        ERR_print_errors_fp(stderr);
        exit(1);
}

/* Verificamos el certificado del servidor */

SSL_CTX_set_verify(ctx,SSL_VERIFY_PEER,NULL);
SSL_CTX_set_verify_depth(ctx,1);
```

En la primera instrucción cargamos el certificado RSA_CLIENT_CA_CERT que será el certificado de la autoridad certificadora que creamos. Con las dos siguientes instrucciones obligamos a que el servidor se autentifique. De esta forma nos aseguramos que el servidor al que conectemos posea un certificado de la autoridad certificadora que hemos especificado. Consiguiendo así, que la conexión se realiza realmente con el servidor deseado y asegurando que no somos víctimas de un ataque de man in the middle.

Lo siguiente que debemos hacer es realizar la conexión con el servidor. Sobre esta conexión crearemos una conexión SSL.

```
/* Creamos un socket TCP */

sock=socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

RETURN_ERR(sock, "socket");

memset(&server_addr,'\0', sizeof(server_addr));
server_addr.sin_family=AF_INET;

server_addr.sin_port=htons(s_port);       /* Puerto del servidor */

server_addr.sin_addr.s_addr=inet_addr(s_ipaddr); /* IP del servidor */

    /* Establecemos una conexion TCP/IP */

err = connect(sock, (struct sockaddr*) &server_addr, sizeof(server_addr));
RETURN_ERR(err, "connect");
```

Como vemos, creamos un socket para una conexión de tipo TCP sobre la dirección IP y el puerto que el cliente introdujo al lanzar el programa. Ahora ya tenemos una conexión TCP al servidor y un contexto configurado para crear una conexión SSL.

```
/* Creamos una estructura SSL */

ssl = SSL_new (ctx);

RETURN_NULL(ssl);

/* Asignamos al socket la estructura SSL */
SSL_set_fd(ssl, sock);

 /* Comenzamos la conexion SSL */
err = SSL_connect(ssl);

RETURN_SSL(err);
```

Creamos nuestra estructura SSL usando la configuración cargada en el contexto y asignamos al socket esa estructura. Tras esto sólo tenemos que realizar la conexión SSL, mediante el método connect. Este método realizará toda la negociación necesaria para establecer la conexión SSL por lo que no tenemos que preocuparnos por nada más. De hecho, si todo ha ido correctamente, ya podríamos intercambiar mensajes con el servidor de forma segura. Pero antes de eso, mostraremos alguna información interesante como el tipo de cifrado o el certificado del servidor.

```
/* Tipo de cifrado usado */
printf ("\nConexion SSL cifrada bajo: %s\n", SSL_get_cipher (ssl));

/* Imprimimos el certificado del servidor */
server_cert = SSL_get_peer_certificate (ssl);

if (server_cert!=NULL){
        printf ("\nCertificado del Servidor:\n");
        str = X509_NAME_oneline(X509_get_subject_name(server_cert),0,0);
        RETURN_NULL(str);
        printf ("\t Servidor: %s\n", str);
        free (str);

        str = X509_NAME_oneline(X509_get_issuer_name(server_cert),0,0);
        RETURN_NULL(str);
        printf ("\t Emitido por: %s\n", str);
        free(str);

        X509_free (server_cert);
```

La información sobre el tipo de cifrado está en la estructura ssl. De hecho toda la información sobre la conexión esta en esta estructura. Para acceder a los datos usaremos métodos ya definidos, lo que hace tremendamente fácil e intuitivo obtener sobre la conexión. De la misma forma podemos obtener el certificado del servidor al que hemos conectado. Este certificado sólo puede estar emitido por nuestra autoridad certificadora, ya que antes lo hemos definido así. Para comprobarlo, además del nombre del certificado, imprimimos el emisor de certificado, de esta forma, el cliente también sabrá que está conectado al servidor correcto.

Mediante los comandos write, y read, podemos intercambiar mensajes con el servidor, pero todavía no hemos realizado la autentificación como cliente. Para completar la autentificación, recordemos que debemos presentar el certificado, que el servidor lo valide, y demostrar poseer la calve privada correspondiente mediante la realización de la firma.

```
printf("\nIniciando autentificacion en el servidor\n");
strcpy(hello,"Iniciar autentificacion Cliente");
err=SSL_write(ssl, hello, strlen(hello));

RETURN_SSL(err);

/* Recivimos reto del servidor */
err = SSL_read(ssl, buf, sizeof(buf)-1);

RETURN_SSL(err);
buf[err] = '\0';

printf("\nreto recivido... Procediendo a firmar el reto con el DNIe\n\n");
```

Mediante la función SSL_write mandamos un mensaje al otro lado de la conexión SSL. En este caso le indicamos que queremos autentificarnos. Tras eso, con el comando SSL_read esperaremos a que el servidor nos responda. El mensaje que recibamos será el reto que deberemos firmar.

Para realizar la firma necesitaremos utilizar las librerías del DNIe. Esta probablemente sea la parte más interesante del código, por lo que la veremos con más detenimiento después.

```
random=buf;

//Firmamos el reto con el DNIe
int res=firmarDNIe(random,&signature);
if(res==0)
        printf("\nExito firmarDNIe\n");
else{
        printf("\nError firmarDNIe\n");
        err = SSL_write(ssl,"ERROR",strlen("ERROR"));
```

```
        goto close;
}

//MANDA LA FIRMA DIGITAL
err = SSL_write(ssl, signature,256);
  RETURN_SSL(err);
//MANDA EL CERTIFICADO
char buffercertif[4096];
FILE *certif;
certif=fopen("/tmp/certif.pem", "r");
fread(buffercertif,sizeof(char),4096,certif);
fclose(certif);
printf("\n");
err = SSL_write(ssl,buffercertif,strlen(buffercertif));
RETURN_SSL(err);
printf("\nfirma y certificado enviados...\n");
```

El reto aleatorio recibido lo guardamos en random y la firma la guardaremos en signature. Estas dos variables las pasaremos a la función firmarDNIe que nos devolverá la firma. Si hubo algún problema indicamos a la otra parte que hubo un error y cerramos la conexión. Para cerrar la conexión iremos a una parte de código etiquetada como close. Más tarde veremos cómo cerrar una conexión. Si por el contrario todo ha ido bien, mandamos la firma al servidor de igual forma que antes. Por último, junto a la firma, debemos mandar el certificado de autentificación del DNIe. La función firmarDNIe también se encargará de extraer el certificado y guardarlo en la carpeta temporal de Linux. Así pues, sólo tenemos que abrir el fichero y enviar el contenido a través de la conexión. Ya hemos enviado todo lo necesario para que el servidor valide nuestra identidad, por lo que sólo nos falta esperar la contestación del servidor.

```
//Esperamos la contestacion del servidor para saber si se completo la autentificacion
int bytes;
bytes=SSL_read(ssl,buf,sizeof(buf));
 if(bytes>0){
        buf[bytes]='\0';
        if(strcmp(buf,"ERROR\0")==0){
                printf("No se pudo completar la autentificacion\n");
                goto close;
        }
}
        else
                goto close;

//Autentificacion exitosa
printf("\nAUTENTIFICACION COMPLETADA\n\n\n");
system("clear");
printf("ya estas conectado al chat\n\n");
```

Con ssl_read, nos quedaremos esperando la respuesta del servidor. Si su respuesta es error, significará que el servidor no pudo completar la autentificación, por lo que avisaremos al cliente y cerraremos la conexión. Si por el contrario, todo ha ido bien ya estaremos autentificados ante el servidor, por lo que ya podremos intercambiar mensajes con los demás usuarios del chat.

Ya hemos visto que el intercambio de mensajes es muy sencillo con ssl_read y ssl_wirte. Sin embargo esto supone un problema, ya que ssl_read es bloqueante. Esto significa que deja el programa a la espera hasta que llega un mensaje, lo que supondría un gran problema para nuestro chat, ya que no podrías mandar ningún mensaje mientras estés esperando una respuesta. La solución más simple es crear un hilo que se quede esperando mensajes del servidor, mientras que otro mande los mensajes que el usuario introduzca por teclado.

```
int pid;
if(pid=fork()){//Proceso padre
	while(1){
		printf ("\n");
		fgets (hello, 80, stdin);
		err=SSL_write(ssl, hello, strlen(hello));

		RETURN_SSL(err);
	}
}
else{//Proceso hijo
	while(1){
		err = SSL_read(ssl, buf, sizeof(buf)-1);
		if(err>0){
			RETURN_SSL(err);
			buf[err] = '\0';
			printf ("%s\n", buf);
		}
		else
			goto close;
	}
}
```

En este caso, el proceso padre enviará los mensajes que el usuario introduzca por teclado mientras que el proceso hijo leerá e imprimirá por pantalla los mensajes que lleguen desde el servidor.

Por último veremos cómo cerrar conexiones correctamente.

```
printf ("\nCerrando conexion\n");
err = SSL_shutdown(ssl);
RETURN_SSL(err);

 /* Terminamos la conexion con el socket*/
err = close(sock);
```

```
RETURN_ERR(err, "close");

/* Liberamos la estructura SSL */
SSL_free(ssl);

      /* Liberamos el contexto */
SSL_CTX_free(ctx);
```

Como podemos ver, ssl ofrece un método para cerrar la conexión, ssl_shutdown. Tras esto deberemos cerrar el socket sobre el que iba nuestra conexión SSL y liberar la memoria destinada para almacenar su estructura y su contexto. Nada complicado.

Con esto habríamos visto el funcionamiento del cliente, pero nos queda la parte más interesante, la firma con el DNIe. Recordemos que hemos definido una función firmarDNIe que recibirá un reto que deberá firmar con la clave privada del certificado de autentificación. Además debemos extraer este certificado y guardarlo en un archivo.

```
int firmarDNIe(unsigned char *random, unsigned char* *signature)
```

Como vemos, esta función recibirá en random el reto a firmar, y guardará la firma en signature que ha sido pasado por referencia. Si la función devuelve un número que nno sea cero es que algo ha salido mal.

```
ctx = PKCS11_CTX_new();

/* Cargamos el Modulo PKCS11 */
rc = PKCS11_CTX_load(ctx,"/usr/lib/opensc-pkcs11.so");
if (rc) {
        fprintf(stderr, "Carga fallida del modulo PKCS#11 ( /usr/lib/opensc-pkcs11.so ):
%s\n",ERR_reason_error_string(ERR_get_error()));
        rc = 1;
        goto nolib;
}
```

Lo primero que hacemos es crear un contexto, y sobre él cargamos el módulo PKCS11, que nos permitirá interactuar con nuestro DNIe.

```
/* Obtenemos informacion de los slots */
rc = PKCS11_enumerate_slots(ctx, &slots, &nslots);
if (rc < 0) {
        fprintf(stderr, "No se encontro ninguna tarjeta\n");
        rc = 2;
        goto noslots;
}
```

Tras esto, buscamos una lista con los lectores de tarjetas que se encuentren conectadas al equipo.

```
/* Usaremos el primer slot con una tarjeta */
slot = PKCS11_find_token(ctx, slots, nslots);
if (!slot || !slot->token) {
        fprintf(stderr, "No se encontro ninguna tarjeta\n");
        rc = 3;
        goto notoken;
}
```

Seleccionaremos el primer lector que tenga una tarjeta insertada. Dado que normalmente no conectamos varias tarjetas al equipo podemos hacer esta simplificación.

```
printf("\n\nTarjeta encontrada:\n");

printf("Manufacturador del Lector.....: %s\n", slot->manufacturer);
printf("Descripcion del Lector.....: %s\n", slot->description);
printf("Nombre de la Tarjeta.....: %s\n", slot->token->label);
printf("Manufacturador de la Tarjeta.....: %s\n", slot->token->manufacturer);
printf("Modelo de la Tarjeta.....: %s\n", slot->token->model);
printf("Numero de serie de la Tarjeta.....: %s\n", slot->token->serialnr);
```

Tras esto, imprimimos toda la información que tenemos sobre el lector y la tarjeta, para que el usuario sepa en todo momento con que lector y tarjeta estamos interactuando. Ahora queremos acceder a las claves y certificados que contiene la tarjeta, pero para ello debemos de saber el código Pin de la tarjeta.

```
/* Hacemos que el password no se vea por pantalla */
if (tcgetattr(0, &old) != 0)
        goto failed;

new = old;
new.c_lflag &= ~ECHO;
if (tcsetattr(0, TCSAFLUSH, &new) != 0)
        goto failed;
```

Dada la importancia del código Pin, es más recomendable, que cuando el usuario lo escriba, el programa no lo muestre por pantalla ya que cualquiera podría verlo. Por ese motivo, deshabilitamos la escritura en pantalla de la entrada estándar, es decir, del teclado.

```
/* leemos el password. */
printf("\nPassword para la tarjeta %.32s: ", slot->token->label);
char *chorra=fgets(password, sizeof(password), stdin);

/* restauramos la terminal */
(void)tcsetattr(0, TCSAFLUSH, &old);
```

```
rc = strlen(password);
if (rc <= 0)
        goto failed;
password[rc-1]=0;
```

Tras realizar este paso, sólo tenemos que pedir al usuario que introduzca el código Pin y guardarlo en una variable, en este caso la variable será password. Una vez que tenemos el código Pin, podemos restaurar la escritura por pantalla del usuario.

```
rc = PKCS11_login(slot, 0, password);
memset(password, 0, strlen(password));
if (rc != 0) {
        fprintf(stderr, "Login en la tarjeta fallido\n");
        goto failed;
}
```

Ya tenemos el código Pin, por lo que podemos hacer login en la tarjeta. Antes de hacer nada más, comprobamos que el login se efectuó correctamente.

```
/* Obtenemos todos los certificados de la tarjeta */
rc = PKCS11_enumerate_certs(slot->token, &certs, &ncerts);
if (rc) {
        fprintf(stderr, "Fallo al enumerar certificados\n");
        goto failed;
}
if (ncerts <= 0) {
        fprintf(stderr, "No se encontraron certificados\n");
        goto failed;
}

/* Usamos el primer certificado: El de autentificacion */
authcert=&certs[0];

printf("\nCertificado Seleccionado: %s\n", authcert->label);
```

Ahora podemos acceder a los certificados del DNIe. Obtenemos una lista de ellos y seleccionamos el certificado de autentificación. Otra opción sería preguntar al usuario que certificado desea usar, pero cómo la única idea de esta aplicación es autentificarse siempre deberá seleccionar el certificado de autentificación. De esta forma evitamos posibles problemas, ya que explicamos, no es para nada recomendable realizar autentificaciones con el certificado de firma digital.

```
authkey = PKCS11_find_key(authcert);
if (!authkey) {
        fprintf(stderr, "No se pudo encontrar la clave para el ceritificado\n");
        goto failed;
}
```

Ya tenemos seleccionado el certificado de autentificación, pero para realizar la firma necesitaremos la clave privada correspondiente a este certificado. Esta clave la guardaremos en authkey.

```
/* Firmamos el reto con la clave privada */
siglen = MAX_SIGSIZE;
(*signature) = malloc(MAX_SIGSIZE);
if (!signature)
        goto failed;
```

Ya hemos dicho que devolveremos la firma en la variable signature. Para poder almacenar esta firma, hacemos una reserva de memoria de 256 bits, que es exactamente lo que ocupa la firma.

```
rc = PKCS11_sign(NID_sha1, random, RANDOM_SIZE, (*signature), &siglen,authkey);
if (rc != 1) {
        fprintf(stderr, "fatal: Fallo al firmar\n");
        goto failed;
}
```

Para poder firmar, pasamos el reto y su tamaño, la variable en donde se guardará la firma, la clave privada para poder firmar, y elegimos el tipo de resumen que se realizará para firmar. En este caso sha1.

```
/* Extraemos la clave publica del certificado */
pubkey = X509_get_pubkey(authcert->x509);
if (pubkey == NULL) {
        fprintf(stderr, "No se pudo extraer la clave publica del certificado\n");
        goto failed;
}
```

Para poder comprobar que la firma es correcta, necesitamos la clave pública correspondiente. La forma más sencilla de conocerla es extraerla del certificado que tenemos seleccionado.

```
/* Verificamos que la firma es correcta */
rc = RSA_verify(NID_sha1, random, RANDOM_SIZE,(*signature), siglen, pubkey-
>pkey.rsa);
if (rc != 1) {
        fprintf(stderr, "fatal: Error verificando la firma\n");
        goto failed;
}
```

Para verificar la firma, la función es muy parecida. Pasaremos el tipo de resumen hecho, el reto y su tamaño, la firma y su tamaño. Lo único que cambia es que ahora queremos verificar y no firmar por lo que pasaremos la clave pública y no la privada. Lógicamente, la validación de la firma se deberá hace en el servidor, y se hará de manera

análoga. El reto ya lo conoce, ya que lo ha creado el mismo, pero deberemos pasarle la firma y el certificado. La firma ya la tenemos, por lo que sólo nos falta guardar el certificado en un fichero.

```
//Imprimimos el certificado a fichero
printf("\nExtrayendo certificado... \n");

FILE *certif;/*El manejador de archivo*/
certif=fopen("/tmp/certif.pem", "w");

if(certif==NULL)
        goto failed;
int sal;
sal=PEM_write_X509(certif,authcert->x509);
fclose(certif);
```

Como ya teníamos el certificado en una variable, las librerías de openSSL nos permiten almacenar este certificado en un archivo en formato PEM sin ninguna complicación. Una vez hecho esto sólo nos resta devolver éxito, en este caso devolver cero.

Ya tenemos nuestro cliente hecho, para compilarlo deberemos linkar todas las librerías que hemos usado. Para mayor comodidad es recomendable hacer un Makefile:

```
CFLAGS=-O2 -g $(shell pkg-config --cflags libp11)
LDFLAGS=$(shell pkg-config --libs libp11)

all: cliente

cliente:
        gcc -lssl -lcrypto -lp11 cliente.c -o CLIENTE

clean:
        rm CLIENTE
```

3.14.4.3. Servidor:

Una vez visto el funcionamiento del cliente, podemos intuir como funcionará el servidor. Esperará conexiones en un puerto, y cuando lleguen se crearán conexiones SSL. Efectivamente es así, pero hay que tener en cuenta varios aspectos que pueden complicar más el desarrollo del programa. Lo primero y más evidente, es que un servidor debe ser capaz de atender a varias conexiones a la vez. La mejor solución es la creación de un hilo nuevo para cada conexión. Pero mirando un poco más allá, nos damos cuenta de que al tratarse de un chat, todos los mensajes que lleguen al servidor deben ser reenviados a todos los usuarios que estén conectados al chat. Esto supone que los hilos, deben comunicarse de alguna manera, cuando llegue un mensaje, para así reenviarlo, pero tenemos que garantizar que ningún mensaje se pierde en caso de que lleguen dos a la vez.

En el cliente vimos la forma más básica de crear un hilo, pero en este caso necesitaríamos señales entre hilos, semáforos y variables compartidas. Dado que el uso conjunto de estas técnicas es bastante frecuente, existen librerías que nos facilitarán enormemente el trabajo. En este caso utilizaremos las librerías pthread. Para comprender mejor el funcionamiento del servidor veremos algunas nociones básicas sobre esta librería.

La creación de un proceso ligero o hilo que ejecuta una determinada función se realiza mediante el método pthread_create. El primer argumento debe ser la dirección de memoria de una variable, donde esta función almacenará el identificador del hilo que se creará. El segundo argumento especificará los atributos de ejecución asociados al nuevo proceso ligero. Si este valor es NULL, se tomarán los atributos por defecto, que incluyen la creación del hilo como no independiente, o joinable. El tercer argumento indica el nombre de la función a ejecutar cuando el proceso ligero comienza su ejecución. Esta función sólo requiere un parámetro que se indica con el cuarto argumento, arg.

A parte de la creación de hilos, tenemos métodos para la realización de operaciones básicas, como esperar la terminación de un hilo, pthread_join o finalizar un hilo, pthread_exit. Sin embargo la mayor ventaja de usar estas librerías viene de poder utilizar métodos para la sincronización entre procesos. Los mecanismos que se utilizan para la sincronización son los mutex (semáforos binarios) y las variables condicionales. Una variable condicional ofrece un mecanismo para que los hilos esperen el cumplimiento de predicados en el que intervengan variables compartidas. Las variables compartidas se deberán proteger con un mutex. Cada vez que un hilo modifica una de estas variables compartidas debe señalizarlo (con pthread_cond_signal) mediante la variable condicional que ha realizado el cambio. Esta señal activará un hilo en espera que se habrá bloqueado usando pthread_cond_wait) que debe comprobar de nuevo si ahora se cumple su predicado. La espera en la variable condicional conlleva la liberación del mutex asociado para que otros hilos puedan modificar las variables compartidas. Cuando se activa un hilo en espera, se vuelve a competir por el mutex.

Para utilizar un mutex, un programa debe declarar una variable de tipo pthread_mutex_t e iniciarla antes de usarla. De forma análoga, para utilizar una variable condicional, es necesario declarar una variable de tipo pthread_cond_t e iniciarla antes de usarla.

Los métodos más importantes para la sincronización de hilos mediante mutex y variables condicionales son: Iniciar o destruir un mutex, pthread_mutex_init o phread_mutex_destroy. Competir por un mutex o liberarlo, pthread_mutex_lock o pthread_mutex_unlock. Iniciar o destruir una variable condicional pthread_cond_init o pthread_cond_destroy. Esperar en una variable condicional, pthread_cond_wait. Esta función suspende al proceso ligero hasta que otro proceso ligero ejecute una operación de señalización sobre la variable condicional pasada como argumento. De forma atómica se libera el mutex pasado como segundo argumento. Cuando el proceso se despierte volverá a competir por el mutex. Señalar una variable condicional, pthread_cond_signal. Se despierta a un proceso suspendido en la variable condicional pasada como argumento(al despertarse tiene que competir por el mutex). No tiene efecto si no hay ningún proceso ligero esperando en la variable condicional.

Una vez tenemos comprendido como realizar la sincronización entre hilos usando la librería pthread, podemos definir mucho mejor como funcionará nuestro servidor. Crearemos un socket a la espera de conexiones, y dejaremos preparado toda la configuración para montar una conexión SSL sobre este socket. Cuando llegue una conexión crearemos un hilo nuevo que atienda a cada cliente mediante el método servidor. Tras realizar la conexión SSL mediante un hilo para cada cliente, enviaremos el reto que el cliente debe firmar. Junto con la firma recogeremos el certificado del cliente. Si no logramos autentificar al cliente con éxito cerraremos esta conexión. En caso contrario, el cliente estará autentificado y ya podrá intercambiar mensajes con los demás clientes. Para el intercambio de mensajes, habremos creado un hilo aparte, llamado responder. Este hilo se quedará a la espera de que llegue un nuevo mensaje y cuando llegue lo reenviará a todos los clientes conectados. Para ello utilizaremos una variable compartida, que guardará el último mensaje, protegiéndolo mediante el uso de un mutex. Para avisar al hilo responder que ha habido cambios utilizaremos las variables condicionales no lleno y no vacio.

De igual forma que antes, para que quede más claro, explicaremos las partes más importantes del código.

```
if(count!= 2){
        printf("Uso : %s <puerto>\n", strings[0]);
        exit(0);
}
```

A diferencia que en el cliente, sólo nos hace falta un argumento para lanzar el programa. El parámetro que nos pase el usuario, será el puerto en el que el servidor escuchará.

```
//Genero el reto
printf("Generando reto...\n");
/* Longitud del reto */
unsigned short int length = 20;
char reto[20];
srand((unsigned int) time(0) + getpid());
int aux=0;
while(length--) {
        reto[aux]=putchar(rand() % 94 + 33);
        srand(rand());
```

```
        aux++;
}
reto[aux]='\0';
printf("\n");
const char* HTMLecho;
HTMLecho=reto;
```

Lo siguiente que hacemos es generar una frase aleatoria. Esta frase aleatoria será el reto que el cliente deberá firmar para completar la autentificación. Es conveniente que el reto cambie de una sesión a otra ya que la captura de un reto firmado supondría a un atacante poder simular la posesión de una clave privada que en realidad no posee.

```
SSL_library_init();
OpenSSL_add_all_algorithms();
SSL_load_error_strings();
method=SSLv23_method();
ctx=SSL_CTX_new(method);
if(ctx==NULL){
        ERR_print_errors_fp(stderr);
          abort();
}
```

De manera análoga a como lo realizamos en el cliente, cargamos las librerías de SSL e iniciamos un nuevo contexto. La única diferencia es que a la hora de elegir el método dejaremos que el cliente pueda conectar con la versión 2 o 3 de SSL. Este método elegirá siempre la última versión disponible por el cliente, pero no obligará a que todos los clientes tengan la última versión. Se trata de poder atender a cuantos más clientes podamos, pero de una manera segura, por lo que no se permitirán conexiones de la versión 1 de SSL.

```
if(SSL_CTX_use_certificate_file(ctx,CertFile,SSL_FILETYPE_PEM)<=0) {
        printf("No se pudo usar el archivo de certificado\n");
        ERR_print_errors_fp(stderr);
        abort();
} else{
        printf("\nCargando certificado del Servidor...\n");
}
if(SSL_CTX_use_PrivateKey_file(ctx,KeyFile,SSL_FILETYPE_PEM)<=0){
        printf("No se pudo usar el archivo de llave\n");
        ERR_print_errors_fp(stderr);
        abort();
} else{
        printf("\nCargando llave privada del servidor...\n");
}
if(!SSL_CTX_check_private_key(ctx)){
        printf("No se pudo usar el archivo de certificado\n");
        fprintf(stderr,"La llave primaria no corresponde al certificado publico\n");
        abort();
}
```

Como el servidor debe autentificarse con el cliente, es necesario que carguemos tanto el certificado como la clave privada correspondiente que creamos en apartados anteriores. Tras cargar certificado y clave, lo que hacemos antes de empezar a aceptar conexiones es comprobar que el certificado seleccionado corresponde con la calve privada.

Esto permitirá a los clientes establecer una conexión cifrada con nuestro servidor, pero además estarán seguros de que los mensajes cifrados van directamente al ordenador del servidor del chat, eliminando cualquier posible ataque de man in th middle. Para usar la clave y el certificado es necesario conocer la PEM phrase, esto es, la contraseña que introdujimos al crearlos. De este modo, nos aseguramos que únicamente nosotros podemos lanzar el servidor. Para que el cliente pueda confiar en nuestro certificado debe poseer el certificado de la autoridad certificadora que lo firmo. Como ya vimos, este es el único certificado que cargamos en el cliente.

```
int sd;
struct sockaddr_in addr;
sd=socket(PF_INET,SOCK_STREAM,0);
bzero(&addr,sizeof(addr));
addr.sin_family=AF_INET;
addr.sin_port=htons(port);
addr.sin_addr.s_addr=INADDR_ANY;
if(bind(sd,(struct sockaddr*)&addr,sizeof(addr))!=0){
        perror("Error al ligar el socket");
        abort();
}
if(listen(sd,10)!=0){
        perror("Error en el listen");
        abort();
}
```

Ahora que ya tenemos configurado correctamente las librerías de SSL, es el momento de crear un socket a la escucha. Para hacer esto sólo necesitamos conocer en que puerto debemos escuchar, pero como ya hemos visto ese parámetro nos lo pasará el usuario al lanzar el programa. Tras esto ya estaremos esperando conexiones de los clientes.

Antes de lanzarnos a ver cómo atender a los clientes veremos un par de detalles importantes.

```
pthread_t res;
pthread_mutex_init(&mutex,NULL);
pthread_cond_init(&no_lleno,NULL);
pthread_cond_init(&no_vacio,NULL);
pthread_create(&res,NULL,responder,NULL);
```

Lo primero que hacemos es crear un semáforo o mutex, para garantizar el acceso seguro y ordenado al último mensaje que llego al servidor. Además creamos dos variables condicionales para poder dormir y despertar los hilos según tengamos o no un mensaje nuevo. Tras esto creamos un hilo que se encargará de reenviar los mensajes a todos los clientes. Este nuevo hilo lanzará el método responder. Tanto este método como el uso del

semáforo y las variables condicionales lo veremos más adelante, pero es importante saber que ya están iniciadas.

Cómo ya vimos, todo lo que necesitamos saber sobre una conexión de SSL está almacenado en una estructura ssl. Así pues, para poder acceder rápidamente a cualquier conexión crearemos una tabla que contendrá todas las conexiones que en ese momento mantengamos. Si no hay ninguna conexión, el valor en la tabla será NULL, por lo que tenemos que tener en cuenta esto a la hora de cerrar conexiones. Para iniciar la tabla como todavía no hay conexiones, lógicamente toda la tabla valdrá NULL.

```
int i;
for(i=0;i<NUM_THREADS;i++){
        tabla_ssl[i]=NULL;
}
```

Ahora veamos cómo hacer para que cada petición sea atendida por un hilo distinto, estableciendo una conexión SSL. El siguiente código irá dentro de un bucle infinito, ya que el servidor estará siempre esperando conexiones.

```
struct sockaddr_in addr;
SSL *ssl;
int len=sizeof(addr);
int client=accept(server,(struct sockaddr*)&addr,&len);
```

Si llega una conexión a nuestro socket, la aceptamos, pero todavía no hemos creado un hilo para atender esta conexión, y ni siquiera hemos asigando este socket a una conexión ssl.

```
ssl=SSL_new(ctx);
// Se crea contexto
SSL_set_fd(ssl,client);
```

Lo primero que hacemos es crear el contexto de una conexión SSL y asignárselo a la conexión TCP que tenemos establecida. Ahora deberíamos crear un hilo para atender dicha conexión y establecer definitivamente la conexión SSL

```
// Se da numero de pthread
thread_data_array[t].thread_id=t;
// Se le da el id del thread al thread
thread_data_array[t].message=ssl;
// Se le da el asigna el contexto de la conexion al thread
thread_data_array[t].reto=(char *)HTMLecho;
```

Pero este nuevo hilo necesitará conocer algunas variables, como el reto, el estado de la conexión ssl o un número que lo identifique. Estos datos los cargamos en la estructura que se pasará como argumento a la hora de crear la conexión.

Ya hemos dicho que en una tabla íbamos a guardar la información sobre todas las conexiones ssl que tengamos establecidas, por lo que ya podemos guardar esta nueva conexión en su lugar correspondiente en la tabla.

```
tabla_ssl[t]=ssl;
```

Con todo esto hecho, ya podemos lanzar el nuevo hilo que atenderá a este cliente. Este nuevo hilo se iniciará en el método Servidor y podrá acceder a los datos que le hemos pasado en la estructura que acabamos de rellenar.

```
rc=pthread_create(&threads[t],NULL,Servidor,(void *)&thread_data_array[t]);
```

Ya hemos visto el funcionamiento general del programa, pero nos queda aún por ver, los dos métodos más importantes. El responder, que se encargará de reenviar todos los mensajes que envíen los clientes, y el servidor que se encargará de autentificar a los clientes y recibir sus mensajes. Quizá el método más importante de todo el programa sea el servidor, por lo que comenzaremos con él.

```
my_data=(struct thread_data*)threadarg;
taskid=my_data->thread_id;
SSL *ssl=my_data->message;
HTMLecho=my_data->reto;
```

Lo primero que hacemos es recoger los datos que hemos mandado por la estructura. Estos datos son: el reto a firmar por el cliente, la estructura para la conexión SSL y un identificador del hilo.

```
if(SSL_accept(ssl)==FAIL){
        printf("Hubo un error en el SSL accept\n");
        SSL_write(ssl,reply,strlen(reply));
        ERR_print_errors_fp(stderr);
} else{
```

Tras esto sólo tenemos que hacer un SSL_accept, con lo que ya tendremos la conexión ssl establecida en el hilo. Ahora ya podemos intercambiar mensajes de forma segura con las funciones read y write, como hicimos en el cliente. Ahora es el momento de realizar la autentificación del cliente, teniendo siempre presente el orden en el que se intercambian los mensajes.

```
bytes=SSL_read(ssl,buf,sizeof(buf));
if(bytes>0){
        buf[bytes]='\0';
        printf("\nIniciando una autentificacion\n");
        if(strcmp(buf,"ERROR\0")==0){
                printf("No se pudo completar la autentificacion del cliente\n");
                goto close;
        }
```

El primer mensaje es la petición del reto por parte del cliente, sólo tenemos que tener en cuenta, que si el cliente envía error significará que algo ha ido mal, por lo que cerraremos la conexión.

Para enviar el reto no tendremos ningún problema ya que es uno de los datos que hemos pasado al hilo. Lo enviaremos como siempre con la función SSL_write.

```
sprintf(reply,HTMLecho,buf);
printf("\nenviando reto al cliente...");
//Enviamos el reto al cliente
SSL_write(ssl,reply,20);
```

Ahora tendremos que recoger la firma del reto y el certificado del cliente. Hay que tener en cuenta, que el cliente nos enviará los datos en este orden y no en ningún otro.

```
//recogemos la firma del reto
char buffirma[4096];
bytes=SSL_read(ssl,buffirma,sizeof(buffirma)-1);
if(bytes>0){
	buffirma[bytes]='\0';
	printf("\nrecogiendo firma del cliente...");
	if(strcmp(buffirma,"ERROR\0")==0){
		printf("NO se pudo completar la autentificacion del cliente\n");
		goto close;
	}
}
else
	goto close;
```

Recogemos el reto de manera análoga, pero siempre comprobando que el cliente no envía error. Ahora haremos lo mismo para recoger el certificado.

```
//recogemos el certificado
bytes=SSL_read(ssl,buf,sizeof(buf));
if(bytes>0){
	buf[bytes]='\0';
	printf("\nrecogiendo certificado del cliente...");
	if(strcmp(buffirma,"ERROR\0")==0){
		printf("NO se pudo completar la autentificacion del cliente\n");
		goto close;
	}
}
else
	goto close;
```

Pero para el uso de certificados, es más que conveniente guardar su contenido en un fichero, ya que casi todas las funciones de manejo de certificados los leen desde fichero.

```
FILE *fs;
char filename[25];
```

```
sprintf(filename,"%s%d","/tmp/certifSERVER",taskid);
fs=fopen(filename,"w+");
if(fs==NULL){
        printf("ERROR creando fichero\n");
        goto close;
}
fwrite(buf,1,4096,fs);
fclose(fs);
```

Para evitar posibles colisiones, el archivo será guardado con el nombre certifServer, seguido del número del identificador del hilo, que lógicamente será distinto para cada hilo. Ya tenemos el reto, la firma y su certificado correspondiente, por lo que ya podemos realizar las operaciones para la autentificación del cliente.

Lo primero que haremos será extraer la clave pública del certificado. Para ello, primero salvaremos en una variable X509 todo el certificado.

```
if(fopen(filename,"r")==NULL)
        close;
X509 *certificado;
certificado=X509_new();
if (PEM_read_X509(fs,&certificado,NULL,NULL)==NULL){
        printf("\nFALLO leyendo certificado\n");
        goto close;
}
fclose(fs);
```

Una vez que tenemos el certificado en la variable x509, podemos extraer la clave pública que contiene el certificado. De momento sólo nos interesa la clave pública, ya que es lo único que necesitamos para verificar la validez de la firma.

```
//Extraemos la clve publica del certificado
EVP_PKEY *pubkey = NULL;
pubkey = X509_get_pubkey(certificado);
if (pubkey == NULL) {
        printf("\nFALLO extrayendo clave publica\n");
        goto close;
}
```

Con la posesión de la clave pública, tenemos todo lo necesario para verificar la firma. La función para verificar la firma es exactamente igual que el que realizamos en el cliente. Es decir, pasaremos como argumentos el tipo de resumen hash realizado, el reto y su tamaño, la firma y su tamaño, y por supuesto la clave pública.

```
if(RSA_verify(NID_sha1, HTMLecho, 20,buffirma,256,pubkey->pkey.rsa)!=1){
        printf("Error verificando firma\n");
        goto close;
}
if (pubkey != NULL)
EVP_PKEY_free(pubkey);
```

```
printf("\nLa firma del reto es correcta.");
```

Ahora ya sabemos que la firma es correcta, es decir, que ha sido realizada con la clave privada correspondiente al certificado que hemos recibido. Sin embargo no hemos comprobado que este certificado sea un certificado válido correspondiente a un DNIe.

```
line=X509_NAME_oneline(X509_get_issuer_name(certificado),0,0);
if(strcmp(line,"/C=ES/O=DIRECCION GENERAL DE LA POLICIA/OU=DNIE/CN=AC
DNIE 003")==0){
        strcpy(issuer,"ACDNIE003-SHA1.pem");
}

else if(strcmp(line,"/C=ES/O=DIRECCION GENERAL DE LA
POLICIA/OU=DNIE/CN=AC DNIE 002")==0){
        strcpy(issuer,"ACDNIE002-SHA1.pem");
}

else if(strcmp(line,"/C=ES/O=DIRECCION GENERAL DE LA
POLICIA/OU=DNIE/CN=AC DNIE 001")==0){
        strcpy(issuer,"ACDNIE001-SHA1.pem");
}

else{
        printf("CA desconocida\n");
        goto close;
}
```

Imprimimos el nombre del emisor del certificado, que deberá ser una subordinad de la dirección general de la policía; la 001, la 002 o la 003. Pero sólo con esto no sirve para garantizar que este certificado sea válido. Si queremos garantizar que el certificado sea válido, y que no está revocado, tendremos que hacer una consulta OCSP para comprobar su estado.

La realización de un cliente OCSP se sale un poco del objetivo de este proyecto, por lo que sólo lanzaremos un comando para realizar la consulta y recogeremos los resultados. Nos basta con saber que hay que hacer dicha consulta para que la autentificación sea completa.

```
sprintf(comando,"openssl ocsp -CAfile acraiz-dnie.cer -issuer %s  -cert %s -url
http://ocsp.dnie.es/ -noverify > %saux", issuer,filename,filename);

system(comando);
```

Como vemos, para la realización de la consulta necesitaremos, aparte del certificado del usuario, el certificado raíz de la autoridad certificadora y el certificado de la autoridad subordinada que haya emitido el certificado. El resultado lo volcaremos a un fichero. Para evitar las posibles colisiones, usaremos el mismo nombre que utilizamos para guardar el fichero, pero añadiéndole aux. De esta forma nos garantizamos que todas las consultas son guardadas en ficheros distintos, y que podemos distinguir que fichero corresponde a cada petición.

```
sprintf(filenameaux, "%saux",filename);
faux=fopen(filenameaux,"r");
if(faux==NULL){
        printf("Errror durante el proceso de validacion\n");
        goto close;
}
char cadena[20];
char cadena1[20];
fscanf(faux,"%s %s\\n",cadena,cadena1);
printf("%s %s\n",cadena, cadena1);
```

Abrimos el fichero, y leemos su contenido. La respuesta al certificado sólo puede ser: 'good' en caso de que todo este correcto, 'revoked' en caso de que el certificado esté revocado, o 'unknown' si el certificado es desconocido.

Lógicamente, si la respuesta obtenida no es good informaremos al cliente de que no se pudo completar la autentificación y cerraremos la conexión.

```
if(strcmp(cadena1,"good")!=0){
        printf("Error validando certificado\n");
        HTMLecho="Error";
        sprintf(reply,HTMLecho,buf);
        SSL_write(ssl,reply,20);
        goto close;
}
```

Si por el contrario, la respuesta ha sido good, avisaremos al cliente que ha sido autentificado con éxito en nuestro servidor, y que ya puede intercambiar mensajes con el resto de clientes del chat.

```
HTMLecho="OK";
sprintf(reply,HTMLecho,buf);
SSL_write(ssl,reply,20);
printf("El estado del certificado es bueno.\n\n");

printf("\nCLIENTE AUTENTIFICADO CON EXITO\n");
MostrarCertificados(certificado,taskid);//Imprimimos el certificado del cliente
```

Para el intercambio de mensajes, tenemos que tener en cuenta, que el servidor recibe los mensajes, y debe reenviarlos a todo los usuarios. Por este motivo, dejamos en un bucle infinito un ssl_read, esperando los mensajes del cliente correspondiente al hilo.

```
while(1){
        bytes=SSL_read(ssl,buf,sizeof(buf));
        if(bytes>0){
                buf[bytes]=0;
                MostrarCertificados(certificado,taskid);
                printf("Mensaje cliente [%d]: %s\n", taskid,buf);
```

Ya tenemos el mensaje, y poseemos el certificado del cliente por lo que podemos extraer sin problemas el nombre y apellidos del cliente que envía el mensaje. Toda esta información debe ser reenviada a todos los clientes del chat de forma ordenada y evitando posibles colisiones.

```
pthread_mutex_lock(&mutex);
if(n_elementos==1)
        pthread_cond_wait(&no_lleno,&mutex);
```

Guardaremos el último mensaje recibido en un buffer común. Para acceder a él, tendremos que competir por el mutex. Si hay un elemento, esperaremos hasta que el buffer no esté lleno.

```
n_elementos=1;

if(certificado!=NULL){
        line=X509_NAME_oneline(X509_get_subject_name(certificado),0,0);
        line1=strstr(line,"CN=");
        line=strstr(line1,"=");
        line2=strtok(line,"(");
        line=strtok(line2,"=");
        sprintf(buffercomun,"> %s: %s",line,buf);
        }

pthread_cond_signal(&no_vacio);
pthread_mutex_unlock(&mutex);
```

Si por el contrario, no hay ningún mensaje guardado, indicaremos que hay uno, que es el que vamos a guardar nosotros. Del certificado extraemos el nombre y apellidos del cliente, y junto con el mensaje lo guardamos en el buffer común. El contenido del buffer común será el último mensaje junto con el nombre y apellidos de quien lo envía. Por tanto, este contenido será el que se deberá reenviar a todos los usuarios. Para ello enviamos la señal de que el buffer no está vacío y liberamos el semáforo. Esta señal será recogida por el método responder que se encargará de enviar el buffer común a todos los clientes.

Antes de ir con el método responder, veremos que hacer para cerrar una conexión, ya que lógicamente los usuarios se desconectan y tenemos que tener esto en cuenta para no tratar de enviar ningún mensaje a un cliente que está desconectado.

```
close:
        printf("\nCliente %d desconectado\n", taskid);
        tabla_ssl[taskid]=NULL;
        sd=SSL_get_fd(ssl);
        SSL_free(ssl);
        close(sd);
```

Como vemos, además de cerrar la conexión SSL y liberar las estructuras, tenemos que indicar en la tabla de conexiones que ya no existe, por lo que ponemos a NULL el valor correspondiente a su identificador en la tabla.

Por último, veamos el método responder, que se encargará de enviar los mensajes a todos los clientes.

```
while(1){
	pthread_mutex_lock(&mutex);
	if(n_elementos==0)
		pthread_cond_wait(&no_vacio,&mutex);

	for(i=0;i<NUM_THREADS;i++){
		if(tabla_ssl[i]!=NULL)
			SSL_write(tabla_ssl[i],buffercomun,strlen(buffercomun));
		}

	n_elementos=0;
	pthread_cond_signal(&no_lleno);
	pthread_mutex_unlock(&mutex);
}
```

El funcionamiento del método es bastante sencillo. Nos quedaremos a la espera de que el buffer no este vacío. Cuando esto ocurra, enviaremos a todos los clientes que tengamos en la tabla de conexiones, el contenido del buffer. Tras esto, indicamos nuevamente que no hay elementos en el buffer, poniendo el número de elementos a cero y enviando la señal para avisar que el buffer ya no está lleno. Tras esto liberamos el semáforo y volveremos a esperar que llegue un nuevo mensaje.

Al igual que hicimos con el cliente, para facilitar la compilación de nuestro código fuente utilizaremos un Makefile con el siguiente contenido.

```
CFLAGS=-O2 -g $(shell pkg-config --cflags libp11)
LDFLAGS=$(shell pkg-config --libs libp11)

all: servidor

servidor:
	gcc -lssl -lcrypto -lpthread servidor.c -o SERVIDOR
clean:
	rm SERVIDOR
```

El desarrollo de este chat nos ha permitido conocer las librerías necesarias para poder desarrollar aplicaciones que interactúen con el DNIe. Además hemos afianzado nuestros conocimientos sobre la autentificación y sobre el protocolo SSL, lo que nos permitirá comprender mejor cómo funciona la página web que desarrollaremos a continuación.

3.14.4.4. Ejecución

A continuación mostraremos un pequeño ejemplo de la ejecución tanto del cliente como del servidor. Lógicamente, primero debemos lanzar el servidor, que en este caso estará a la escucha en el puerto 5555. En este ejemplo se autentificará a dos clientes, en este caso será el mismo cliente, ya que sólo disponemos de un DNIe.

./SERVIDOR 5555

Generando reto...
JIaN_X*gu|qb6}D:N96@

Cargando certificado del Servidor...
Enter PEM pass phrase:

Cargando llave privada del servidor...

Certificado y Llave cargados correctamente.

ESPERANDO CONEXIONES

Iniciando una autentificacion

enviando reto al cliente...
recogiendo firma del cliente...
recogiendo certificado del cliente...
La firma del reto es correcta.
Validando certificado...
/tmp/certifSERVER1: good
El estado del certificado es bueno.

CLIENTE AUTENTIFICADO CON EXITO

Certificados del Cliente[1]:
Certificado:
/C=ES/serialNumber=72708599X/SN=ESTEBAN/GN=XABIER/CN=ESTEBAN LOZANO, XABIER (AUTENTICACI\xC3\x93N)
Emisor: /C=ES/O=DIRECCION GENERAL DE LA POLICIA/OU=DNIE/CN=AC DNIE 003

Certificados del Cliente[1]:
Certificado:
/C=ES/serialNumber=72708599X/SN=ESTEBAN/GN=XABIER/CN=ESTEBAN LOZANO, XABIER (AUTENTICACI\xC3\x93N)
Emisor: /C=ES/O=DIRECCION GENERAL DE LA POLICIA/OU=DNIE/CN=AC DNIE 003

Mensaje cliente [1]: hola

Iniciando una autentificacion

enviando reto al cliente...
recogiendo firma del cliente...
recogiendo certificado del cliente...
La firma del reto es correcta.
Validando certificado...
/tmp/certifSERVER2: good
El estado del certificado es bueno.

CLIENTE AUTENTIFICADO CON EXITO

Certificados del Cliente[2]:
Certificado:
/C=ES/serialNumber=72708599X/SN=ESTEBAN/GN=XABIER/CN=ESTEBAN LOZANO, XABIER (AUTENTICACI\xC3\x93N)
Emisor: /C=ES/O=DIRECCION GENERAL DE LA POLICIA/OU=DNIE/CN=AC DNIE 003

Certificados del Cliente[2]:
Certificado:
/C=ES/serialNumber=72708599X/SN=ESTEBAN/GN=XABIER/CN=ESTEBAN LOZANO, XABIER (AUTENTICACI\xC3\x93N)
Emisor: /C=ES/O=DIRECCION GENERAL DE LA POLICIA/OU=DNIE/CN=AC DNIE 003
Mensaje cliente [2]: que tal?

Cliente 1 desconectado

Como vemos, el servidor debe controlar la autentificación, conexión, desconexión e intercambio de mensajes entre los clientes. De esta forma destacamos aún más un tema en el que no hemos incidido: Los algoritmos criptográficos protegen el canal por el que enviamos la información, pero no sirven de nada si la seguridad de los equipos ya está comprometida. Es importante tener esto en cuenta a la hora de usar nuestro DNIe, ya que por muy seguro que sean los algoritmos de cifrado, no van a servir de mucho si un atacante consigue tener un control total sobre tu equipo. Dico esto, veamos un ejemplo de conexión de un cliente.

./CLIENTE 192.168.2.3 5555

Conexion SSL cifrada bajo: AES256-SHA

Certificado del Servidor:
Servidor: /C=ES/ST=Navarra/L=Pamplona/O=Proyecto Xabi/OU=Servidor/CN=Servidor/emailAddress=servidor@servidor.com
Emitido por: /C=ES/ST=Navarra/L=Pamplona/O=Proyecto Xabi/OU=CA Autoridad Certificadora/CN=Autoridad Certificadora/emailAddress=autoridad@certificadora.com

Iniciando autentificacion en el servidor

reto recivido... Procediendo a firmar el reto con el DNIe

Tarjeta encontrada:
Manufacturador del Lector.....: OpenSC (www.opensc-project.org)
Descripcion del Lector.....: C3PO LTC31 00 00
Nombre de la Tarjeta.....: DNI electrÃ³nico (PIN1)
Manufacturador de la Tarjeta.....: DGP-FNMT
Modelo de la Tarjeta.....: PKCS#15
Numero de serie de la Tarjeta.....: 0606A2526026A0

Password para la tarjeta DNI electronico (PIN1):
Certificado Seleccionado: CertAutenticacion

Extrayendo certificado...

Exito firmarDNIe

firma y certificado enviados...

AUTENTIFICACION COMPLETADA

ya estas conectado al chat

hola

> ESTEBAN LOZANO, XABIER : hola

> ESTEBAN LOZANO, XABIER : que tal?

Como vimos, sólo es necesario introducir el Pin y no seleccionar el certificado. El propio programa se encarga de seleccionar el certificado de autentificación que es el apropiado para este tipo de aplicaciones. Para poder confiar en este servidor imprimimos su

certificado, que obligatoriamente tiene que estar firmado por la autoridad certificadora que hemos creado. De esta forma evitamos posibles ataques de man in the middle.

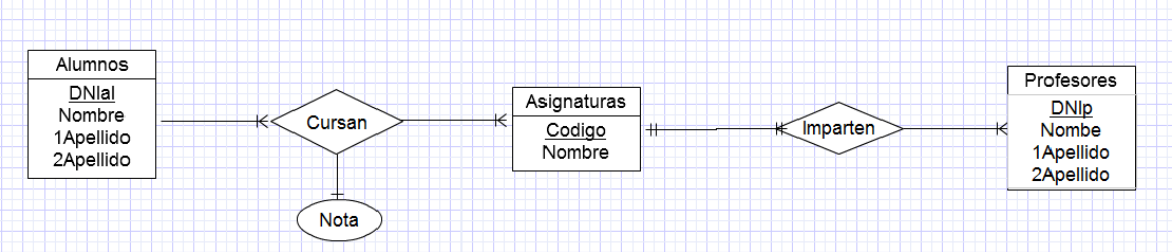
3.15. Aplicación 2: Página web

Ahora vamos a realizar una página web en la que los clientes tengan que autentificarse mediante el uso del DNIe, asegurando que la persona es en realidad quien dice ser, y una vez hecho, se les mostrará una información personalizada. Además daremos la posibilidad de que los usuarios puedan firmar formulario en línea, permitiéndonos estar seguros que esa firma sólo la puede haber realizado esa persona, y que lo firmado no ha sido modificado en ningún momento.

Más concretamente, realizaremos un pequeño sistema de publicación y visionado de calificaciones para los profesores y alumnos de la universidad. Todo usuario deberá autentificarse mediante el DNIe, tras lo que verificaremos si esta persona imparte o cursa alguna asignatura. En caso de ser alumno, se le permitirá el visionado de todas sus notas de una sola vez, pero lógicamente no podrá ver notas que no sean suyas. Si por el contrario es profesor, podrá ver la lista completa de notas de todas las asignaturas que imparte. Además podrá publicar notas para su asignatura, pero para ello deberá firmar las notas mediante el DNIe, asegurando que además de proceder del profesor correspondiente, estas notas no han sido modificadas por ningún agente externo.

3.15.1. Base de datos

Antes de comenzar a desarrollar parece claro que vamos a necesitar una base de datos para almacenar de forma ordenada, los datos de los alumnos, profesores y asignaturas de la universidad. Para ello, haremos un pequeño diagrama de entidad relación, que nos ayudará a realizar la base de datos.



De esta forma, podemos identificar rápidamente las entidades, sus atributos y relaciones que tendrá nuestra base de datos. Como gestor de bases de datos usaremos mySQL, un servidor bastante rápido y popular, que se suele utilizar en los sitios de hosting.

Dado que este programa es muy popular, no tendremos ningún problema a la hora de instalarlo. Bastará con buscar e instalar los paquetes mysql-server y mysql-client. En el momento de la instalación se pedirá la contraseña de administrador.

Una vez hayamos instalado el programa, accederemos a él mediante el comando mysql –h localhost –u root –p, dónde localhost es la dirección de tu host y root el nombre de usuario. El usuario root o administrador podrá crear nuevos usuarios y otorgarles diferentes permisos para el acceso y modificación de la base de datos. Al tratarse de una base de datos muy pequeña y a la que sólo accederemos nosotros nos bastará con el usuario root, pero tendríamos que tener esto en cuenta para proyectos más grandes.

Una vez logueados, podremos comenzar a crear nuestra base de datos.

```
create database proyecto;
use proyecto;
```

Tras crear la base de datos, la seleccionamos, y comenzaremos a añadir las tablas necesarias.

```
CREATE TABLE `Alumnos` (
`DNIal` VARCHAR( 9 ) NOT NULL ,
`Nombre` VARCHAR( 15 ) NOT NULL ,
`1Apellido` VARCHAR( 15 ) NOT NULL ,
`2Apellido` VARCHAR( 15 ) NOT NULL ,
constraint pk_alumnos primary key(DNIal)
) ENGINE = MYISAM;
```

La tabla alumnos tendrá contendrá el DNI nombre y apellidos de los alumnos, y lógicamente su clave primaria será el DNI ya que no puede haber dos personas que tengan el mismo DNI.

```
CREATE TABLE `Profesores` (
`DNIp` VARCHAR( 9 ) NOT NULL ,
`Nombre` VARCHAR( 15 ) NOT NULL ,
`1Apellido` VARCHAR( 15 ) NOT NULL ,
`2Apellido` VARCHAR( 15 ) NOT NULL ,
constraint pk_profesores primary key(DNIp)
) ENGINE = MYISAM;
```

Exactamente lo mismo haremos para los profesores.

```
CREATE TABLE `Asignaturas` (
`Codigo` INT( 5 ) NOT NULL,
`Nombre` VARCHAR( 50 ) NOT NULL ,
`DNIp` VARCHAR( 15 ) ,
constraint pk_asignaturas primary key(Codigo)
) ENGINE = MYISAM;
```

En cuanto a las asignaturas, almacenaremos el código que las identifica, que será la calve primaria, el nombre de la asignatura y el DNI del profesor que la imparte.

```
ALTER TABLE `Asignaturas` ADD CONSTRAINT fk_asignaturas FOREIGN
KEY(DNIp) REFERENCES Profesores(DNIp) ON DELETE SET NULL;
```

El DNI del profesor que imparte la asignatura será una clave externa de la tabla profesores. En caso de que borremos a este profesor de la base de datos dejaremos este campo vacío, pero no eliminaremos la asignatura.

```
CREATE TABLE `Cursan` (
`DNIal` VARCHAR( 9 ) NOT NULL ,
`Codigo` INT( 5 ) NOT NULL,
`Nota` DECIMAL(3,1),
constraint pk_cursan primary key(DNIal,Codigo)
) ENGINE = MYISAM;
```

En esta última tabla almacenaremos las notas de cada alumno para cada asignatura, por lo tanto la clave primaria será tanto el DNI del alumno, como el código de la asignatura.

```
ALTER TABLE `Cursan` ADD CONSTRAINT fk_cursan1 FOREIGN KEY(DNIal)
REFERENCES Alumnos(DNIal) ON DELETE CASCADE;
ALTER TABLE `Cursan` ADD CONSTRAINT fk_cursan1 FOREIGN KEY(Codigo)
REFERENCES Asignaturas(Codigo) ON DELETE CASCADE;
```

Al igual que antes, el DNI del alumno y el código de la asignatura serán claves externas, pero en este caso, el borrado de un alumno o de una asignatura acarrearía el

borrado de sus notas correspondientes. No tendría ningún sentido guardar notas de alumnos o de asignaturas que ya no existen.

Ya tenemos todas las tablas creadas, ahora ya podemos introducir valores en nuestra base de datos. Para introducir un alumno haremos:

```
insert into Alumnos values ("72708599X","Xabier","Esteban","Lozano");
```

y para borrarlo:

```
delete from Alumnos where DNIal='72708599X';
```

Para la inserción y borrado de profesores procederemos igual:

```
insert into Profesores values ("72708599X","Xabier","Esteban","Lozano");
delete from Profesores where DNIp='72708599X';
```

Una vez tengamos todos los profesores y alumnos en la base de datos, podremos introducir las asignaturas y las notas:

```
insert into Asignaturas values (55006,"Programacion","72708599X");
insert into Cursan values ("72708599X",55001,NULL);
```

Para la eliminación completa de la base de taso bastará con hacer:

```
drop database proyecto;
```

3.15.2. Servidor web

Ahora ya tenemos nuestra base de datos lista, pero todavía no vamos a empezar a programar el código fuente de nuestra página. Antes deberemos elegir un servidor de páginas web y configurarlo correctamente.

Para incorporar la autentificación y validación de los certificados del DNIe tenemos dos alternativas: Realizarla a través de un Applet de Java o haciendo uso de la autentificación mutua que ofrece el protocolo SSL. La principal diferencia entre uno y otro radica en la transparencia de cara al usuario. Si se hace uso del protocolo SSL, el propio navegador es quien solicita al usuario el certificado y realiza la autentificación, por lo que no es necesario cargar ningún software extra. En cambio, si se opta por el uso de un Applet, el usuario debe indicar a la máquina virtual de Java que confía en el software para que pueda acceder al lector y realizar la autentificación. Obviamente, es mucha más transparente la solución ofrecida por el protocolo SSL, ya que el usuario sólo deberá introducir el código Pin y seleccionar su certificado, por lo que será la que utilizaremos. Hay que tener en cuenta que la implementación del protocolo SSL la ofrece el servidor web donde se ejecuta la aplicación, por lo que se deberá adaptar cada servidor web con una correcta configuración.

El protocolo SSL es muy utilizado en la actualidad, por lo que todos los navegadores lo incorporan. Los servidores web más extendidos en la actualidad son Apache HTTP server, Micrsoft IIs y Apache Tomcat. Este último lo descartamos ya que no vamos a desarrollar nuestra web en java. Cómo estamos haciendo todo el desarrollo en Linux, usaremos el servidor Apache HTTP server, que a día de hoy es con mucho, el servidor web más utilizado.

Para instalar apache en nuestro sistema, bastará con instalar el paquete apache2. A la misma vez podemos instalar el paquete php5, ya que lo necesitaremos para desarrollar en php. Una vez tenemos todo instalado, hay que activar el módulo SSL del servidor. Para ello utilizaremos el comando a2enmod mod_ssl.

Apache escucha conexiones por el puerto 80, que es el destinado para conexiones HTTP, pero ahora también queremos que se puedan establecer conexiones HTTPS, por lo que deberemos indicarlo en el archivo de configuración de apache mediante la directiva Listen 443, que es por defecto el puerto del protocolo HTTPS.

Una vez configurado el puerto, deberemos crear un host virtual que resuelva las conexiones que lleguen al puerto 443. Este host virtual será diferente al que escucha en el puerto 80, por lo que podernos considerar que la web se dividirá en dos partes. En una, a la escucha en el puerto 80, no será necesario ningún tipo de autentificación; mientras que en la otra parte será obligatoria la autentificación mediante el uso del DNIe.

Podemos usar de base la configuración por defecto que trae el host virtual para atender peticiones en el puerto 443. Con simplemente cambiar el nombre o el directorio donde se encuentra la carpeta raíz del entorno web. Con esa configuración ya podríamos suministrar webs a las peticiones que llegasen al puerto 443, pero no estaríamos estableciendo una conexión segura. Para activar el protocolo SSL en nuestro host virtual deberíamos añadir las líneas:

```
SSLEngine on #Se activa el protocolo SSL
SSLCipherSuite HIGH:MEDIUM:-SSLv2 #se indica la configuración de seguridad
SSLCertificateFile "/etc/certs/apache.pem" #se suministra la ruta al certificado
SSLCertificateKeyFile "/etc/certs/apache.pem" #se suministra la ruta a la clave privada
```

Como se puede apreciar, indicamos al servidor que active el protocolo SSL y el grado de seguridad que soporta, en este caso media y alta, rechazando el protocolo SSL versión dos, que debido a unas vulnerabilidades descubiertas ya no se considera seguro. Lógicamente, para poder establecer una conexión segura, debemos indicarle al servidor que clave y certificado usar. La creación de claves y certificados ya ha sido tratada anteriormente, por lo que no nos entretendremos más en esto.

Llegados a este punto, nuestro servidor web ya es capaz de establecer conexiones seguras por el puerto 443 con el cliente o navegador. Por último, debemos obligar a que el cliente también presente su certificado. Esto se hace con la instrucción SSLVerifyClient, que puede tomar los siguientes valores: none, para no solicitar; optional en caso de ser opcional; o requiere, que será el que usemos, para que sea obligatorio la presentación de un certificado válido. Lógicamente, el uso de esta instrucción conlleva indicar en que autoridad certificadora se confía.

```
# Certificado raiz DNIe
SSLCACertificateFile "/etc/certs/ac_raiz_dnie.crt"
# El cliente debe autenticarse obligatoriamente con el certificado
SSLVerifyClient require
# Nivel máximo de profundidad (según infraestructura actual, 2)
SSLVerifyDepth 2
```

El certificado raíz del DNIe se puede descargar desde la página oficial del DNIe, pero vine en formato DER y Apache espera un formato PEM, por lo que deberemos convertirlo.

```
openssl x509 -in ac_raiz_dnie.crt -inform DER -out ac_raiz_dnie.crt -outform PEM
```

Además del certificado de la autoridad certificador hemos indicado la profundidad máxima de CA entre el certificado que se confía, el raíz, y el que el usuario presenta. Este valor será dos, ya que todo DNIe ha sido emitido por una autoridad certificadora intermedia que depende de la raíz, como ya vimos anteriormente.

Con todo esto, ya somos capaces de autentificar a los usuarios con el DNIe, pero no podemos asegurar que el certificado que presente no esté revocado. Para comprobar esto tendremos que hacer una consulta OCSP, de igual manera que hicimos en el chat. Esta consulta no está integrada en los servidores web por lo que es labor del desarrollador. Más tarde vernos como realizarla, pero ahora nos queda una última cosa que hacer. La instrucción SSLOptions + StdEnvVars + ExportCertData exportará los datos del usuario a PHP.

3.15.3. Código

Como ya hemos dicho, la web se dividirá en dos partes, una zona pública, a la que todo el mundo puede acceder, y una zona privada, para la que será necesario el uso del DNIe. La zona pública constará de una única página, que dará la bienvenida y mostrará un enlace para acceder a la parte privada. Aún así, es interesante destacar algún detalle sobre la página.

```
<!DOCTYPE html PUBLIC "âˆ’//W3C//DTD XHTML 1.0 Transitional //EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1âˆ’transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta name="author" content="Xabier Esteban Lozano">
<meta name="description" content="PFC: Visionado y publicacion de calificaciones de la
UPNA a travÃ©s del DNIe">
<meta name="keywords" content="DNIe UPNA calificaciones">
<title>Sistema de Calificaciones UPNA</title>
```

Definimos as cabeceras con el nombre del autor, una breve descripción y una lista de palabras clave. Por último damos título a la página.

```
<style type="text/css">
body {
      font-family: Georgia, "Times New Roman", serif;
         font-size:14pt;
      color: #000000;
      background-color: #FFFFFF;
         text-align: center;
    }
table {
         border-style: none;
}
</style>
```

Ahora definimos el estilo de la página. Esto nos permitirá cambiar el estilo sin tener que preocuparnos por el contenido. Una forma de hacerlo es definir una hoja de estilo en un archivo externo que todas las páginas compartan. Sin embargo, definiremos la hoja de estilo en cada página, ya que esto permitirá que con mayor facilidad se cambie el estilo de una única página. Por ejemplo, nos gustaría que las calificaciones de los alumnos se publicasen con una fuente más grande, pues sólo tendríamos que ir a la página donde se muestran las notas de los alumnos y modificar un valor en la hoja de estilo. Esto modificará el estilo de la página de los alumnos, pero no modificará el resto.

```
</head>
<body>
<table width="100%">
<tr>
<td><img src="./logo.upna.640.gif"> </td>
<td> <H1>  Sistema de calificaciones  </H1></td>
<td><img src="./DNIe.gif" height="100" width="100"> </td>
</tr>
</table>
<HR>
```

En cuanto al contenido de la página, siempre tendremos una tabla como título, mostrando el titulo de la página y los logos del DNIe y de la UPNA.

```
<br>
<br>
<br>
<br>
<br>
Bienvenido al sistema de calificaciones de la UPNA<br>
Para poder entrar es necesario el uso del DNIe<br><br>
<A HREF="https://192.168.2.3"> Identificarse con el DNIe</A>
<br>
<br>
<br>
<br>
<br>
<br>
```

La única parte que cambiará en las páginas será esta, por lo que de ahora en adelante, sólo nos centraremos en esta parte del código, y obviaremos el resto, ya que será igual. En este caso, sólo damos la bienvenida al usuario y le mostramos un enlace para acceder a la zona privada, recordando que es necesario el uso del DNIe. Como ya hemos visto, sólo nos preocupamos del contenido de la página, ya que el estilo lo hemos definido.

```
<br>
<HR>
<table align="right">
<tr>
<td> Esta web es un PFC de Xabier Esteban Lozano </td>
</tr>
</table>

</body>
</html>
```

Debajo del contenido de la página, mostraremos que esta página es un proyecto final de carrera. Con cerrar todas las etiquetas, acabamos la página de inicio.

En la primera página de la zona privada, tenemos que completar la autentificación. Hasta ahora, sabemos que sólo han podido llegar a esta página usuarios que posean un certificado y su calve correspondiente, firmado por la autoridad certificadora del DNIe y que su plazo no hay expirado. Sin embargo, es posible que este certificado este revocado. Por ejemplo, un DNIe robado, sigue teniendo sus certificados firmados por la dirección general de la policía, pero cuando el dueño denuncie su desaparición, estos certificados serán revocados. Por este motivo, para realizar una autentificación completa, debemos realizar la consulta OCSP para comprobar el estado exacto del certificado.

En cuanto al código, dejaremos todas las páginas como la anterior, y solo cambiaremos la parte central del contenido. En este caso:

```
<?php

if(isset($_SESSION["dni"])){
        echo "Tu DNIe ha sido autentificado con exito";
        echo "<br>";
        echo "<br>";
        echo "<a href='https://192.168.2.3/principal.php'>Continuar</a>";
}
```

Si la variable de sesión dni esta ya iniciada, significará que el usuario ya ha sido autentificado con éxito y que ha vuelto a esta página. Para evitar una consulta OCSP innecesaria, sólo mostraremos un enlace para ir a la página principal.

En caso contrario, significará que el usuario aún no ha sido autentificado, por lo que esto será lo primero que debamos hacer.

```
else{

$dir = '/tmp/';
$RootCA = './acraiz-dnie.cer';
$OCSPUrl = 'http://ocsp.dnie.es/';

$a = rand(1000,99999);
file_put_contents($dir.$a.'cert_i.pem', $_SERVER['SSL_CLIENT_CERT_CHAIN_0']); //
Certificado del emisor del certificado

file_put_contents($dir.$a.'cert_c.pem', $_SERVER['SSL_CLIENT_CERT']); // Certificado
de autentificacion del cliente

$output = shell_exec('openssl ocsp -CAfile '.$RootCA.' -issuer '.$dir.$a.'cert_i.pem -cert
'.$dir.$a.'cert_c.pem -url '.$OCSPUrl);
```

Guardamos el certificado del cliente y el certificado de la autoridad que ha emitido este certificado en sendos archivos y hacemos la consulta OCSP.

```
$output2 = preg_split('/[\r\n]/', $output);
```

```
$output3 = preg_split('/: /', $output2[0]);
$ocsp = $output3[1];

if($ocsp!="good"){
        echo "Lo sentimos pero no se pudo validar tu certificado.";
        unlink($dir.$a.'cert_i.pem');
        unlink($dir.$a.'cert_c.pem');
        echo "<br>";
        echo "<br>";
        echo "<a href='http://192.168.2.3'>Salir</a>";
}
```

Si la respuesta que obtenemos es distinta de 'good', significará que el certificado no es válido, por lo que no podemos autentificar al usuario.

```
else{
        $pieces = explode("/",$_SERVER[SSL_CLIENT_S_DN]);
        $pieces = explode("=",$pieces[2]);
        $_SESSION["dni"]=$pieces[1];
        unlink($dir.$a.'cert_i.pem');
        unlink($dir.$a.'cert_c.pem');

        echo "Tu DNIe ha sido autentificado con exito";
        echo "<br>";
        echo "<br>";
        echo "<a href='https://192.168.2.3/principal.php'>Continuar</a>";
}
}
?>
```

Si por el contrario, la respuesta es buena, guardaremos el DNI del certificado en una variable de sesión, que nos indicará que el usuario ha sido autentificado con éxito. De esta forma sólo realizamos la consulta OCSP una vez, y no estamos haciendo todo el rato la misma consulta. Tras autentificar al usuario, le mostramos un enlace para la página principal.

La página principal, se encargará de identificar si el DNI corresponde con un profesor o alumno de la universidad, y de ser así, mostrarle un menú con sus opciones. Para ello necesitaremos iniciar la sesión y conectar con la base de datos. Como esta parte la tendremos que realizar siempre de ahora en adelante, añadiremos este código al principio de cada página.

```
<?php
        session_start();
```

Primero iniciamos la sesión, para recuperar los valores que ya tengamos inicados de entes, en este caso el DNI.

```
function conectarBaseDatos() {
        if(!($link=mysql_connect("localhost","root","xabi"))){
```

```
		echo "Error conectando a la base de datos.";
		exit();
	}
	if(!mysql_select_db("proyecto",$link)) {
		echo "Error seleccionando la base de datos.";
		exit();
	}
	return $link;
}
```

Tras esto definimos una función que realiza la conexión a nuestra base de datos, de esta forma, con sólo llamar a esta función nos evitamos tener que estar escribiendo todo el rato este mismo código, a demás de ganar en legibilidad.

```
if(!isset($_SESSION["dni"])){
	echo "No se pudo completar la autentificacion";
	echo "<br>";
	echo "<a href='http://192.168.2.3'>Salir</a>";

}
```

Lo primero que haremos será verificar que el usuario haya sido ya autentificado, de no ser así, sólo le mostraremos un enlace a la salida.

```
echo "Hola $_SERVER[SSL_CLIENT_S_DN_G] $_SERVER[SSL_CLIENT_S_DN_S], Bienvenido.";
echo "<br>";
echo "<br>";
echo "<br>";
$link=conectarBaseDatos();
$query=sprintf("SELECT * FROM Alumnos WHERE DNIal='%s'",mysql_real_escape_string($_SESSION["dni"]));
$result=mysql_query($query);
```

Si ya ha sido autentificado, conectaremos con la base de datos y consultaremos los datos del alumno con ese DNI.

```
if(mysql_num_rows($result)==0){
	$query=sprintf("SELECT * FROM Profesores WHERE DNIp='%s'",mysql_real_escape_string($_SESSION["dni"]));
	$result = mysql_query($query);
```

Si no obtenemos resultados, significará que no se trata de un alumno de la universidad, por lo que comprobaremos si es un profesor, de la mima forma que antes.

```
if(mysql_num_rows($result)==0){
	echo "Usted no esta mutriculado en la UPNA y tampoco imparte clase";
	echo "<br>";
	echo "<br>";
```

```
        echo "<a href='http://192.168.2.3'>Salir</a>";
}
```

Si tampoco obtenemos resultados, significará que el usuario no es ni alumno ni profesor de la universidad, por lo que lo único que mostramos es un enlace a la salida.

Si se trata de un profesor de la universidad, debemos darle la posibilidad de consultar las notas de las asignaturas que imparte, o también la posibilidad de publicar notas.

```
echo "Eres profesor de la Universidad publica de Navarra";
$_SESSION["tipo"]="profesor";
echo "<br>";
echo "<br>";
```

En la variable de sesión tipo, guardaremos que tipo de usuario es el cliente, es decir, si el usuario es profesor o alumno de la universidad. De esta forma, con comprobar esto una sola vez será suficiente, y evitaremos consultas redundantes a la base de datos.

```
$query1=sprintf("SELECT * FROM Asignaturas WHERE
DNIp='%s'",mysql_real_escape_string($_SESSION["dni"]));
$result1=mysql_query($query1);
if(mysql_num_rows($result1)==0){
        echo "<br>";
        echo "Ahora mismo no imparte ninguna clase";
        echo "<br>";
        echo "<a href='https://192.168.2.3/salir.php'>Salir</a>";
}
```

Tras esto, consultamos en la base de datos, que asignaturas imparte en este momento. Lógicamente, si no está impartiendo ninguna, no podremos decirle que consulte sus notas o que publique nuevas.

```
echo "Para consultar las notas de sus asignaturas siga el siguinete enlace";
echo "<br>";
echo "<br>";
echo "<a href='https://192.168.2.3/profesores.php'>Consultar notas</a>";
echo "<br>";
echo "<br>";
```

Primero le mostramos un enlace a la página de ‘profesores.php’. En esta página se visualizarán todas las notas de todas las asignaturas que el profesor imparte.

```
echo "<br>";
echo "<br>";
echo "Para publicar notas selecciona la asignatura deseada";
echo "<br>";
echo "<br>";
echo "<FORM method=post action='https://192.168.2.3/publicar.php'>";
echo "<TABLE BORDER=0 align='center'>";
```

```
echo "<tr>";
echo "<td>";
echo "<SELECT name='codigo'>";
while ($row1 = mysql_fetch_assoc($result1)) {
        echo "<OPTION VALUE=".$row1['Codigo'].">".$row1['Nombre']."</OPTION>";
}
echo "</SELECT>";
echo "</tr>";
echo "</td>";
echo "<tr>";
echo "<td>";
echo "<INPUT type='submit' value='Publicar Notas'>";
echo "</tr>";
echo "</td>";
echo "</FORM>";
echo "</table>";
echo "<br>";
```

Además de visualizar las notas, deberemos darle la opción de publicar notas, nuevas. En este caso mostraremos una lista con las asignaturas que imparte, para que seleccione una y se dirija a la página 'publicar.php' que veremos más tarde.

```
echo "<br>";
echo "<br>";
echo "<a href='https://192.168.2.3/salir.php'>Salir</a>";
```

Por último damos la oportunidad de salir de nuestra página al profesor.

Si por el contrario, nos encontramos ante un alumno de la universidad, sólo podemos darle las opciones de salir y de consultar sus notas. Pero al igual que realizamos antes con los profesores, en la variable de sesión tipo guardamos que se trata de un alumno.

```
echo "Eres alumno de la Universidad publica de Navarra";
echo "<br>";
echo "Para ver sus calificaciones siga el siguiente enlace";
echo "<br>";
echo "<br>";
$_SESSION["tipo"]="alumno";
echo "<a href='https://192.168.2.3/alumnos.php'>Ver Notas</a>";
echo " o ";
echo "<a href='https://192.168.2.3/salir.php'>Salir</a>";
```

Estamos viendo, que todos los enlaces de salir dirigen a la página 'salir.php', que tiene el siguiente contenido.

```
<?php
        session_start();
        if (ini_get("session.use_cookies")) {
                $params = session_get_cookie_params();
```

```
                setcookie(session_name(), '', time() - 42000,$params["path"],
$params["domain"],$params["secure"], $params["httponly"]);
        }
        session_destroy();
        echo "<script >";
        echo "location.href='http://192.168.2.3/';";
        echo "</script>";
?>
```

Simplemente destruimos la sesión y eliminamos las cookies Tras esto redirigimos al cliente a la página de presentación de nuestro proyecto. Como vemos, de esta forma nos ahorramos algunas líneas de código cada vez que queremos dar al usuario la opción de salir.

A continuación veremos la página 'alumnos.php' que mostrará las calificaciones del alumno en todas sus asignaturas:

```
if(!isset($_SESSION["dni"]) || $_SESSION["tipo"]!="alumno"){
        echo "<script >";
        echo "location.href='https://192.168.2.3/principal.php';";
        echo "</script>";
}
```

Como siempre, lo primero que hacemos es verificar que el usuario ha sido autentificado con éxito y que se trata de un alumno. Esta información debería estar en las variables de sesión, y si no es así, no podemos dejarle entrar y lo mandamos a la página principal.

```
echo "Hola $_SERVER[SSL_CLIENT_S_DN_G] $_SERVER[SSL_CLIENT_S_DN_S],
Estas son tus calificaciones";
echo "<br>";
echo "<br>";
echo "<br>";
echo "<br>";
echo "<br>";
$link=conectarBaseDatos();
$query=sprintf("SELECT * FROM Cursan WHERE
DNIal='%s'",mysql_real_escape_string($_SESSION["dni"]));
$result=mysql_query($query);
```

Conectamos a la base de datos, y sacamos toda la información de la tabla cursan cuyo campo DNIal coincida con el DNI del usuario. De esta forma obtendremos el código de la asignatura y la nota correspondiente de todas las asignaturas que actualmente esté cursando el alumno.

A continuación construiremos una tabla que contendrá la asignatura y la nota, para ello, sólo tenemos que recorrer el resultado que hemos recogido de realizar la consulta. Pero pese a que las asignaturas se identifiquen por un código único, es poco probable que el alumno conozca las asignaturas por su código, por lo que para cada código de asignatura consultaremos el nombre concreto de esta asignatura.

```
if (!$result) {
	$message  = 'Invalid query: ' . mysql_error() . "\n";
	$message .= 'Whole query: ' . $query;
	die($message);
}
echo "<table border='0' align='center'>";
echo "<tr bgcolor='#FFF100'>";
echo "<td>";
echo "Codigo";
echo "</td>";
echo "<td>";
echo "Asignatura";
echo "</td>";
echo "<td>";
echo "NOTA";
echo "</td>";
while ($row = mysql_fetch_assoc($result)) {
	echo "<tr>";
	echo "<td>";
	echo $row['Codigo'];
	echo "</td>";


	$query1=sprintf("SELECT Nombre FROM Asignaturas WHERE
Codigo='%s'",mysql_real_escape_string($row['Codigo']));
	$result1=mysql_query($query1);
	if (!$result1) {
		$message  = 'Invalid query: ' . mysql_error() . "\n";
		$message .= 'Whole query: ' . $query1;
		die($message);
	}
	while ($row1 = mysql_fetch_assoc($result1)) {
		echo "<td align='left'>";
		echo $row1['Nombre'];
		echo "</td>";
	}
	if(!isset($row['Nota'])){
		echo "<td>";
		echo "No presentado";
		echo "</td>";
	}
	else if($row['Nota']>=5){
		echo "<td bgcolor='#00FFAD'>";
		echo $row['Nota'];
		echo "</td>";
		echo "</tr>";
	}
	else{
		echo "<td bgcolor='#FF6600'>";
		echo $row['Nota'];
```

```
                echo "</td>";
                echo "</tr>";
        }
        }
        echo "</table>";
        echo "<br>";
```

Para una mejor visualización, cambiaremos el color de fondo de la tabla. Si la asignatura está suspendida, el color será rojo y si esta aprobada verde.

```
        echo "<br>";
        echo "<a href='https://192.168.2.3/principal.php'>Volver</a>";
        echo "  o  ";
        echo "<a href='https://192.168.2.3/salir.php'>Salir</a>";
        mysql_free_result($result1);
        mysql_free_result($result);
        mysql_close($link);
}
?>
```

Una vez hecho esto, sólo nos queda mostrar los enlaces de salir o volver a la página principal y cerrar la conexión a la base de datos. Con esta parte concluimos toda la sección de nuestra página web dedicada a los alumnos.

Si el cliente que se conecta a la página web es un profesor, debe poder visionar las calificaciones de las asignaturas que imparte. De esto se encargará la página 'profesores.php', que será muy parecida a la de los alumnos, pero que mostrará las calificaciones de todos los alumnos de cada asignatura.

```
if(!isset($_SESSION["dni"]) || $_SESSION["tipo"]!="profesor"){
        echo "<script >";
        echo "location.href='https://192.168.2.3/principal.php';";
        echo "</script>";
}
```

Como siempre, debemos comprobar que sea un cliente autentificado, y que hayamos verificado que es profesor. De no ser así lo redirigimos a la página principal.

```
echo "Hola $_SERVER[SSL_CLIENT_S_DN_G] $_SERVER[SSL_CLIENT_S_DN_S],
Estas son las calificaciones de las asignaturas que impartes";
echo "<br>";
echo "<br>";
echo "<br>";
echo "<br>";
echo "<br>";
$link=conectarBaseDatos();
$query=sprintf("SELECT Codigo, Nombre FROM Asignaturas WHERE
DNIp='%s'",mysql_real_escape_string($_SESSION["dni"]));
$result=mysql_query($query);
```

```
if (!$result) {
        $message  = 'Invalid query: ' . mysql_error() . "\n";
        $message .= 'Whole query: ' . $query;
        die($message);
}
```

Tras esto consultamos el nombre y el código de todas las asignaturas que imparte el profesor.

```
while ($row = mysql_fetch_assoc($result)) {
        echo "Asignatura: ";
        echo $row['Nombre'];
        echo " (";
        echo $row['Codigo'];
        echo ") ";
        echo "<br>";
        echo "<br>";
        echo "<table border='0' align='center'>";
        echo "<tr bgcolor='#FFF100'>";
        echo "<td>";
        echo "DNI";
        echo "</td>";
        echo "<td>";
        echo "Nombre";
        echo "</td>";
        echo "<td>";
        echo "1 Apellido";
        echo "</td>";
        echo "<td>";
        echo "2 Apellido";
        echo "</td>";
        echo "<td>";
        echo "NOTA";
        echo "</td>";
        echo "</tr>";
        $query1=sprintf("select c.DNIal DNIal, a.Nombre Nombre, a.1Apellido
1Apellido,a.2Apellido 2Apellido, c.Nota Nota from Cursan c, Alumnos a where
c.DNIal=a.DNIal and  Codigo=%s",mysql_real_escape_string($row['Codigo']));
        $result1=mysql_query($query1);
        if (!$result1) {
                $message  = 'Invalid query: ' . mysql_error() . "\n";
                $message .= 'Whole query: ' . $query;
                die($message);
        }
```

Para cada asignatura construiremos una tabla con el dni nombre y apellidos de cada alumno y su nota correspondiente, para lo que hacemos la consulta correspondiente.

```
        while ($row1 = mysql_fetch_assoc($result1)) {
                echo "<tr>";
```

```
                echo "<td align='left'>";
                echo $row1['DNIal'];
                echo "</td>";
                echo "<td align='left'>";
                echo $row1['Nombre'];
                echo "</td>";
                echo "<td align='left'>";
                echo $row1['1Apellido'];
                echo "</td>";
                echo "<td align='left'>";
                echo $row1['2Apellido'];
                echo "</td>";
                if(!isset($row1['Nota'])){
                        echo "<td>";
                        echo "No presentado";
                        echo "</td>";
                }
                else if($row1['Nota']>=5){
                                echo "<td bgcolor='#00FFAD'>";
                                echo $row1['Nota'];
                                echo "</td>";
                                echo "</tr>";
                        }
                        else{
                                echo "<td bgcolor='#FF6600'>";
                                echo $row1['Nota'];
                                echo "</td>";
                                echo "</tr>";
                        }
                }
                echo "</table>";
                echo "<br>";
                echo "<br>";
                echo "<br>";
        }
```

Al igual que hicimos antes, imprimimos todos los resultados que obtenemos de la consulta. Para ayudar a la visualización, los suspensos se marcarán en rojo y los aprobados en verde.

```
        echo "<br>";
        echo "<br>";
        echo "<a href='https://192.168.2.3/principal.php'>Volver</a>";
        echo "  o  ";
        echo "<a href='https://192.168.2.3/salir.php'>Salir</a>";
        mysql_free_result($result1);
        mysql_free_result($result);
        mysql_close($link);
}
?>
```

Tras esto, sólo nos queda mostrar los enlaces de volver o salir y cerrar la conexión con la base de datos. Como hemos visto, esta parte es muy similar a la anterior.

Ahora mismo, los alumnos pueden ver todas sus notas, pero no las de los demás y los profesores también pueden ver todas las notas de las asignaturas que imparten, pero ninguna más. Una vez hecho esto, tenemos que dar al profesor la posibilidad de publicar notas. Para asegurar que estas notas provienen realmente del profesor, y que en ningún momento han sido modificadas, utilizaremos la firma digital del DNIe.

Cuando el profesor seleccione la opción de publicar notas para una determinada asignatura, será dirigido a 'publicar.php'. Esta página mostrará las notas de todos los alumnos de esta asignatura, por lo que es realmente muy parecida a la que acabamos de ver. Sin embargo la diferencia radicará en que junto a la nota, aparecerá una lista con todas las notas para seleccionar una nueva nota. Como casi todo el código es el mismo que antes, veamos sólo la parte interesante.

```
echo "<FORM name='formularioPublicar' method=post
action='https://192.168.2.3/firmar.php'>";
echo "<input type='hidden' name='firmar' value='si'>";
echo "<input type='hidden' name='codigo' value=$codigo>";
```

Como vemos, creamos un formulario, que pasará a 'firmar.php' por el método POST. Como datos ocultos, pasaremos el código de la asignatura, para no perder de qué asignatura son las notas y la variable firma para indicar que realmente procedemos de aquí, y que no hemos ido directamente a la página 'firmar.php'

```
echo "<SELECT name=$dni>";
$notav=$row['Nota'];
if($row['Nota']==NULL){
        echo "<OPTION VALUE=NULL SELECTED>No presentado</OPTION>";
        $aux=0;
        while($aux<=10){
                echo "<OPTION VALUE=$aux>$aux</OPTION>";
                $aux=$aux+0.1;
        }
}
else{
        echo "<OPTION VALUE=$notav SELECTED>$notav</OPTION>";
        echo "<OPTION VALUE=NULL>No presentado</OPTION>";
        $aux=0;
        while($aux<=10){
                echo "<OPTION VALUE=$aux>$aux</OPTION>";
                $aux=$aux+0.1;
        }
}
echo "</SELECT>";
echo "<INPUT type='submit' value='Publicar Notas'>";
echo "</form>";
```

Junto con la nota antigua desplegaremos una lista con todas las posibles notas, pero tendremos cuidado para que la nota antigua esté siempre seleccionada por defecto. La variable tendrá por nombre el DNI del alumno, y por valor la nueva nota, de esta forma no perderemos ninguna información.

La página 'firmar.php', deberá recoger las nuevas notas de los alumnos y solicitar al cliente que firme estas notas con el DNIe. Para interactuar con el DNIe tenemos dos opciones, utilizar un Applet Java para que el cliente realice la firma o utilizar las funciones que tiene el navegador. Al igual que antes, elegiremos la opción más transparente de cara al usuario, es decir, usar las funciones que el navegador tiene para ello. En este caso nos hemos decidido por usar la librería crypto implementada en JavaScript, y más concretamente la función signText. De esta forma, conseguiremos que el usuario pueda firmar notas sin la necesidad de incorporar ningún software extra en el ordenador del usuario. Para nuestro ejemplo sólo trabajaremos con esta librería que da soporte a los navegadores más extendidos como pueden ser Mozilla Firefox, Opera o Netscape. Cabe destacar que el navegador web Internet Explorer no implementa esta función. Para dar soporte a este navegador abría que usar la librería Capicom creada en ActiveX, pero cuyo funcionamiento sería muy parecido, por lo que no nos preocuparemos de ello. Dicho esto, veamos un poco el código de esta página.

Lo primero que hacemos es comprobar que el usuario esta autentificado y que es un profesor. Además, la variable firmar recogida por el método POST, nos dirá que efectivamente, el cliente viene de rellenar el formulario anterior.

```
if(!isset($_SESSION["dni"]) || $_SESSION["tipo"]!="profesor" || $_POST['firmar']!="si"){
        echo "<script >";
        echo "location.href='https://192.168.2.3/principal.php';";
        echo "</script>";
}
```

Tras esto, imprimimos por pantalla como quedarán las nuevas notas, por lo que el código resultante será practicante igual que los dos o tres que acabamos de hacer.

```
echo "Por favor $_SERVER[SSL_CLIENT_S_DN_G]
$_SERVER[SSL_CLIENT_S_DN_S]";
echo "<br>";
echo "Firme las notas para poder publicarlas";
echo "<br>";
echo "<br>";
echo "<br>";
echo "<br>";
$link=conectarBaseDatos();
$query=sprintf("select c.DNIal DNIal, a.Nombre Nombre, a.1Apellido
1Apellido,a.2Apellido 2Apellido, c.Nota Nota from Cursan c, Alumnos a where
c.DNIal=a.DNIal and  Codigo=%s",mysql_real_escape_string($_POST['codigo']));
$result=mysql_query($query);
if (!$result) {
        $message  = 'Invalid query: ' . mysql_error() . "\n";
        $message .= 'Whole query: ' . $query;
        die($message);
```

```
}
```

Recogemos nombre, DNI y apellidos de cada alumno que cursa la asignatura y junto con estos datos imprimimos la nueva nota que va a tener.

```
$codigo=$_POST['codigo'];
while ($row = mysql_fetch_assoc($result)) {
        echo $row['Nombre'];
        echo " ";
        echo $row['1Apellido'];
        echo " ";
        echo $row['2Apellido'];
        echo " de DNI: ";
        echo $row['DNIal'];
        echo " tiene una nota de: ";
        $dni=$row['DNIal'];
        if($_POST[$dni]=="NULL"){
                echo "No presentado";
                $nota="No presentado";
        }
        else{
                echo $_POST[$dni];
                $nota=$_POST[$dni];;
        }
        echo "<br>";
        $linea1=$row['Nombre']." ".$row['1Apellido']." ".$row['2Apellido']." de DNI:
".$row['DNIal']." tiene una nota de: ".$nota."--";
        $linea=$linea.$linea1;
        }
```

De esta forma, el profesor podrá verificar las notas que va a publicar de una manera muy cómoda. Llama la atención que en la variable $linea hemos ido guardando los datos de todos los alumnos con sus notas, ya que esto será lo que firmaremos.

Ahora crearemos un formulario para que el tutor pueda firmar y enviar las notas a la página ‘verificar.php’ que lógicamente se encargará de verificar la firma y publicar finalmente las notas.

```
echo "<FORM name='formularioFirmar' method=post
action='https://192.168.2.3/verificar.php'>";
echo "<input type='hidden' name='verificar' value='si'>";
echo "<input type='hidden' name='codigo' value=$codigo>";
while ($row = mysql_fetch_assoc($result)) {
        $dni=$row['DNIal'];
        echo "<input type='hidden' name=$dni value=$_POST[$dni]>";
}
echo "<input type='hidden' name='firma' value='>";
echo "<input type='text' name='frase' value='$linea' readonly style='visibility: hidden'>";
echo "<br>";
```

```
echo "<input type='submit' value='Firmar' onClick='javascript:signAndSend();'>";
echo "</form>";
```

La variable oculta verificar la usaremos para indicar que realmente procedemos de rellenar ese formulario, y no de ninguna otra parte. Además deberemos pasar el código de la signatura y por supuesto las notas. Para no perder información crearemos una variable con el DNI de cada alumno cuyo valor será la nueva nota. Pero aparte de eso, necesitaremos dos datos más, la frase, que tiene por valor $linea. Es decir, frase será todo el texto que el cliente va a firmar; y firma, que está vacía, pero que más tarde tomará el valor de la firma digital. Ahora ya tenemos todo, por lo que añadimos un botón, que cuando pulsemos lanzará la función signAndSend().

Esta función es la que se encargará de realizar la firma digital y enviar todos los datos.

```
<script language="javascript">
function signAndSend() {

document.formularioFirmar.firma.value=crypto.signText(document.formularioFirmar.frase
.value,"ask");
   document.formularioFirmar.submit();
}
</script>
```

El funcionamiento es bastante claro, la variable firmar recogerá el valor que de cómo resultado usar la función de firma sobre la variable frase, y tras esto enviamos todo a 'verificar.php'. La función crypto.signText se encargará de solicitar el Pin, que el usuario seleccione el certificado y mostrarle lo que está a punto de firmar. Como vemos, usar esta función facilita muchísimo la labor al desarrollador, además de ser mucho más transparente para el cliente por lo que su uso es muy adecuado para el tipo de aplicación que queremos hacer.

Ya sólo nos queda una página por ver, 'verificar.php' que recogerá todos estos datos, validará la firma y publicará los resultados. Para realizar un código más legible definiremos tres funciones:

```
function saveMsg($msg,$file){
        $fp = fopen($file,"w");
        fwrite($fp,$msg);
        fclose($fp);
}
```

Que guardará el mensaje original en un archivo.

```
function saveSig($sig,$file) {
        $data = "-----BEGIN PKCS7-----\n".$sig."\n-----END PKCS7-----\n";
        $fp = fopen($file,"w");
        fwrite($fp,$data);
        fclose($fp);
}
```

Esta función guardará la firma en oro archivo, pero al tratarse de una firma, en formato PKCS7, también tendremos que añadir las cabeceras correspondientes.

Y esta última función que verificará la firma.

```
function verifySig($msgFile,$sigFile,$certFile) {
	$ca_cert="./acraiz-dnie.cer";
	//Extrae certificado de la firma
	exec("openssl pkcs7 -inform PEM -in $sigFile  -outform PEM -out $certFile",
&$salida, &$res);
	if($res!=0) {
		echo "Error en extraccion de certificado\n";
		return $res;
	}
	exec("openssl pkcs7 -in $certFile -print_certs -out $certFile", &$salida, &$res);
	exec("openssl x509 -in $certFile -subject -noout", &$salida, &$res);
```

Lo primero que haremos será extraer el certificado del cliente, que estará incrustado en la firma. Tras esto transformaremos el certificado al formato x509 e imprimaremos el campo subject del certificado.

```
	//Verificamos que corresponde al mismo DNI
	$pieces = explode("/",$salida[0]);
	$pieces = explode("=",$pieces[2]);
	if($pieces[1]!=$_SESSION["dni"]){
		echo "<br>";
		echo "Has usado otro DNIe";
		return -1;
	}
```

Este campo contiene el DNI, por lo que debemos comprobar que el DNI que ha firmado las notas es el mismo que se ha autentificado.

```
	//Verificamos que se trata del certificado de firma
	$pieces = explode("(",$salida[0]);
	$pieces = explode(")",$pieces[1]);
	if($pieces[0]!="FIRMA"){
		echo "<br>";
		echo "Por favor, selecione el certificado de FIRMA";
		return -1;
	}
```

Al tratarse de una firma digital, deberemos comprobar que el certificado es el de firma digital, ya que de no ser así, no sería una firma válida.

```
	//Verificamos la firma
	exec("openssl smime -verify -inform pem -in $sigFile -content $msgFile -CAfile
$ca_cert", &$salida,&$res);
	return $res;
```

```
}
?>
```

Por último verificamos que lo firmado y la firma coinciden y que el certificado está emitido por la autoridad certificadora del DNIe. Con estas funciones ya definidas, nos resultará bastante más fácil el resto de la página.

```
if(!isset($_SESSION["dni"]) || $_SESSION["tipo"]!="profesor" ||
$_POST['verificar']!="si"){
        echo "<script >";
        echo "location.href='https://192.168.2.3/principal.php';";
        echo "</script>";
}
```

Al igual que siempre, lo primero que comprobamos es que hemos autentificado el DNI, que se trata de un profesor de la universidad, y que viene de rellenar el formulario de verificar y no de otra parte.

```
$frase=$_POST["frase"];
$sig= $_POST["firma"];

$msgFile= tempnam("/tmp","Msg");
saveMsg($frase,$msgFile);
$sigFile= tempnam("/tmp/","Sig");
saveSig($sig,$sigFile);
$certFile = tempnam("/tmp/","Cert");
```

Recogemos el mensaje original y la firma, y creamos unos archivos temporales donde guardarlos. Además crearemos un archivo temporal más, que será donde almacenemos el certificado que extraigamos de la firma.

```
$rv = verifySig($msgFile,$sigFile,$certFile);
if ($rv!==0) {
        echo "No se pudo validar la firma<br>\n";
        echo "<br>";
        echo "<br>";
        echo "<a href='https://192.168.2.3/principal.php'>Volver</a>";
        echo "  o  ";
        echo "<a href='https://192.168.2.3/salir.php'>Salir</a>";
}
```

Ahora simplemente tenemos que llamar a la función que hemos declarado y esperar el resultado. Si no recibimos un cero, significará que algo ha ido mal, por lo que se lo comunicaremos al usuario y no modificaremos las notas.

```
echo "La firma es correcta<br>\n";
$link=conectarBaseDatos();
$query=sprintf("select * from Cursan where
Codigo=%s",mysql_real_escape_string($_POST['codigo']));
$result=mysql_query($query);
```

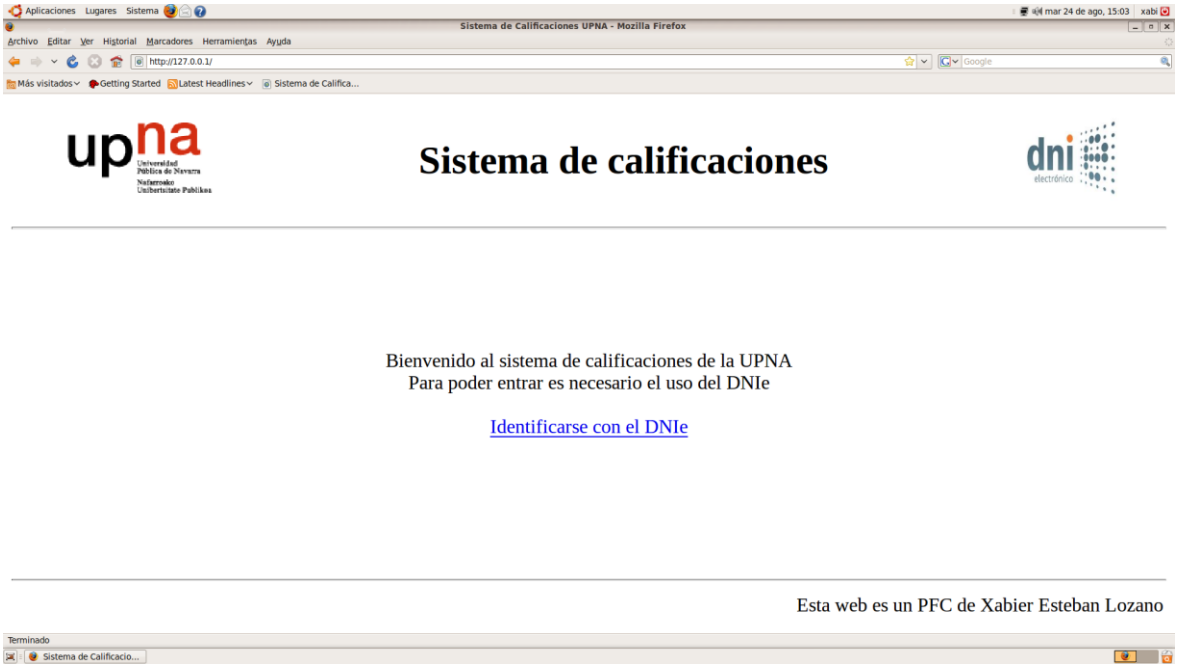
```
if (!$result) {
        $message  = 'Invalid query: ' . mysql_error() . "\n";
        $message .= 'Whole query: ' . $query;
        die($message);
}
while ($row = mysql_fetch_assoc($result)){
        $dni=mysql_real_escape_string($row['DNIal']);
        $codigo=mysql_real_escape_string($_POST['codigo']);
        $nota=mysql_real_escape_string($_POST[$dni]);

        $query=sprintf("UPDATE Cursan SET nota=%s WHERE DNIal='%s' AND
Codigo=%s;",$nota,$dni,$codigo);
        $result1=mysql_query($query);
```

Si por el contrario, la firma ha sido verificada correctamente, ya sólo nos quedará modificar la base de datos. Haremos una consulta para obtener todos los números de DNI de los alumnos que cursan la asignatura correspondiente. En $_POST[$dni] tendremos la nota correspondiente a ese DNI, por lo que ya podemos hacer un update para actualizar nuestra base de datos.

Con esto damos por finalizada la página web, y por tanto la segunda aplicación del proyecto. Como hemos visto, nos ha resultado mucho más cómodo el desarrollo de la página web, ya que el servidor se encarga de hacer automáticamente parte de la autentificación, pero la realización del chat nos ayudó a comprender mejor cómo se debe realizar una autentificación. Además hemos afianzado nuestros conocimientos sobre la firma digital, y sobre los métodos para incorporar la firma digital con un navegador web, sin necesidad de ninguna aplicación extra.

3.15.4. Ejecución:

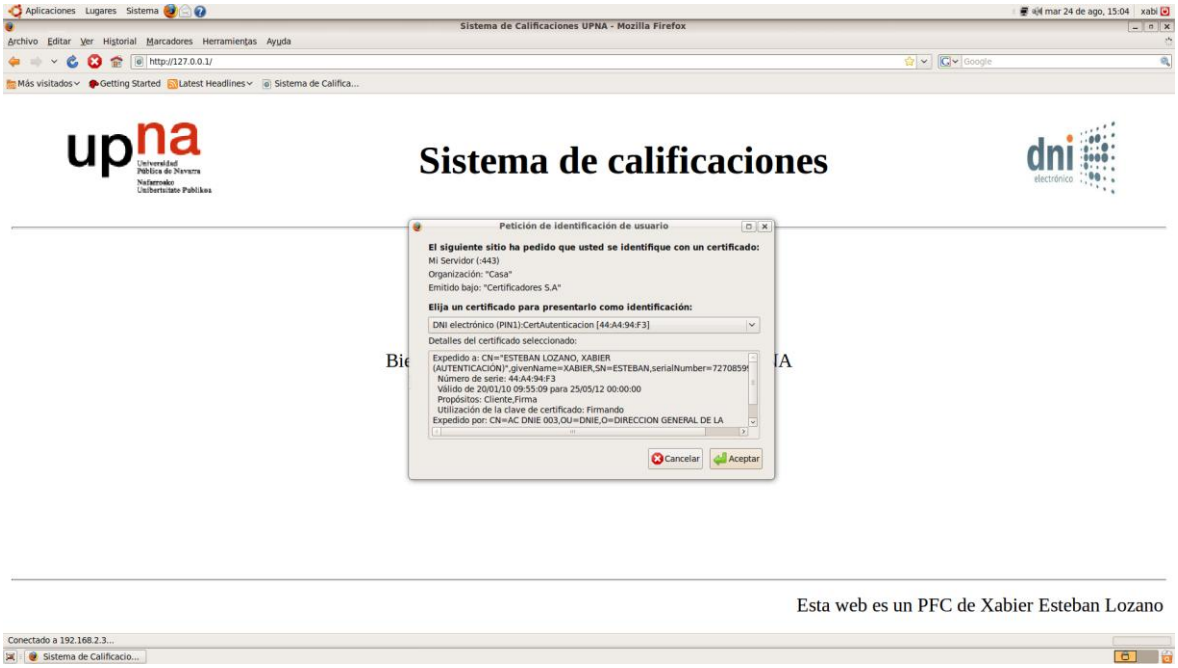
Ahora veremos el resultado de la página. Abrimos el navegador Mozilla y nos dirigimos a nuestra página.



Esta es la parte de presentación, y de momento no hace falta para nada el uso del DNIe. En el centro de la página veremos un enlace hacia la zona privada, avisando que será necesaria la autentificación mediante DNIe.

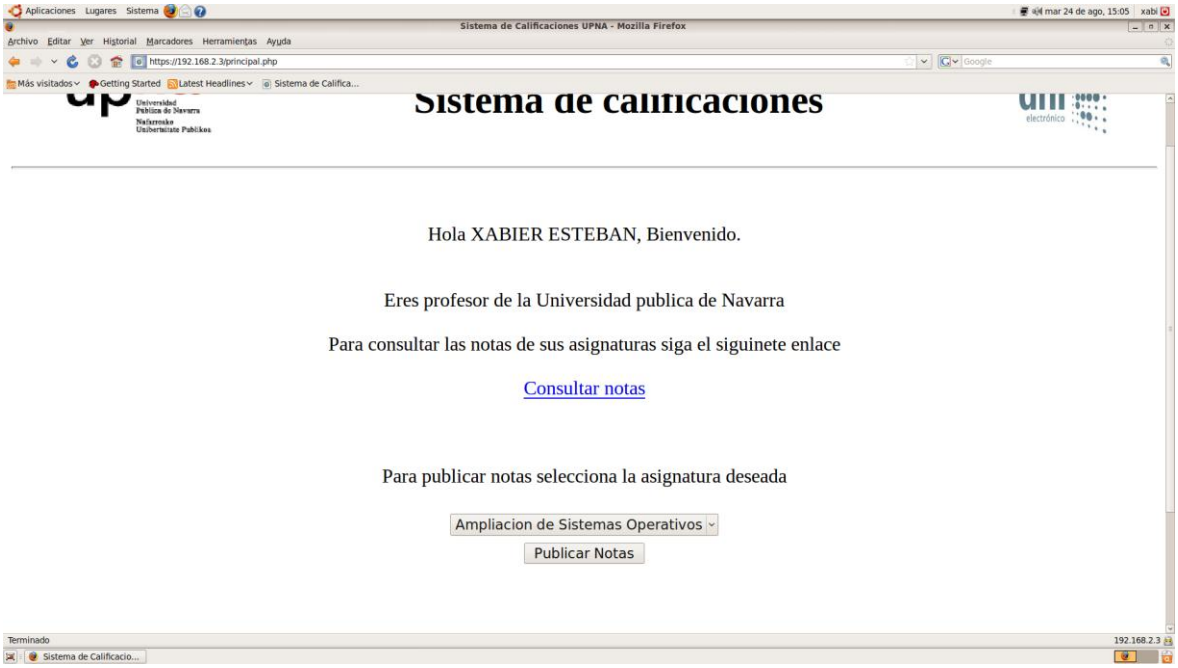
Para que nuestro navegador pueda acceder a los certificados, será necesario introducir el código Pin de nuestro DNIe.

Tras introducir el código Pin, se mostrará un menú desplegable con todos los certificados de los que disponemos. Para realizar una identificación en una web deberemos seleccionar el certificado de autentificación.

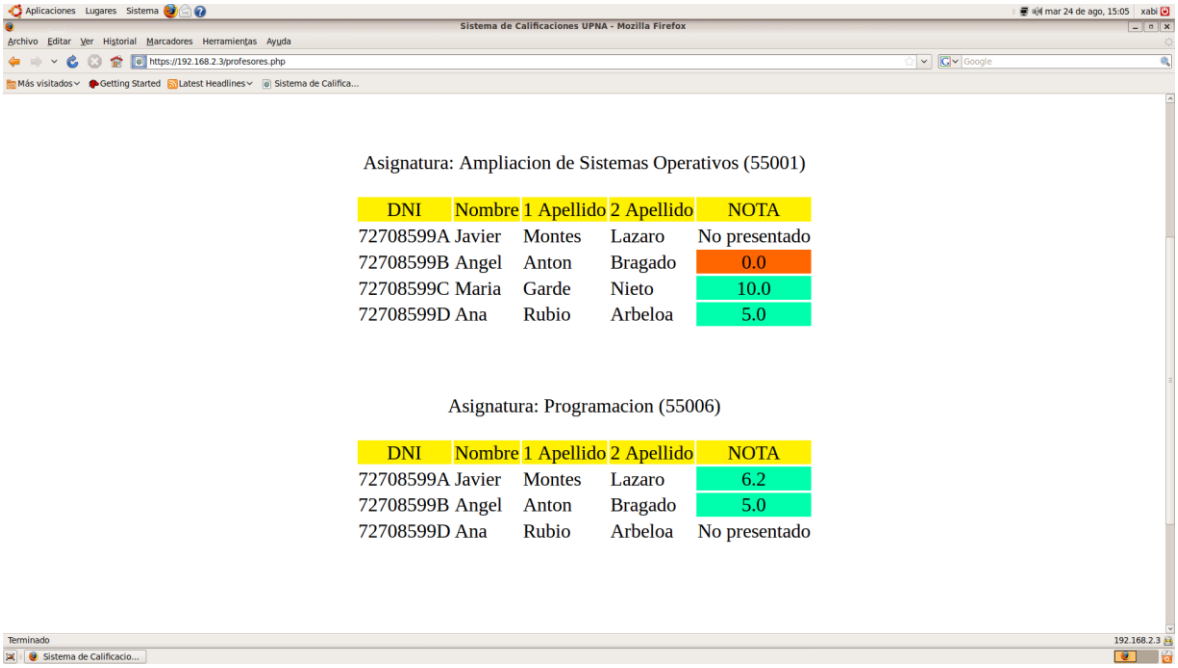


Una vez seleccionado, ya habremos entrado a la página y estaremos autentificados. Como el certificado que creamos para el servidor está firmado por nosotros mismos deberemos avisar al navegador que confíe en él, si es la primera vez que entramos en el sitio.

Como vemos, se nos indicará si se pudo realizar la autentificación con éxito. En caso de haber podido, veremos un enlace hacía el menú principal. En el menú principal se mostrará la posibilidad de ver tus notas, si eres alumno, o de publicar o consultar notas si eres profesor. En este caso supondremos que somos profesores.

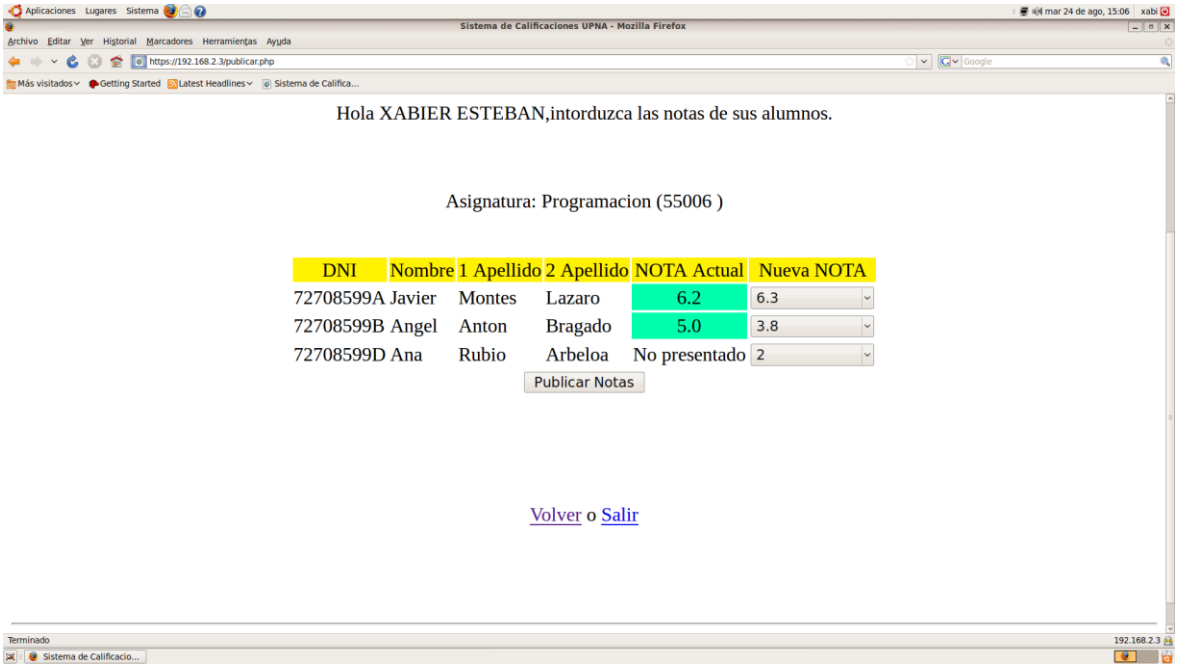


Si pulsamos en consultar notas, podremos ver todas las notas de las asignaturas que impartimos, pero sólo de esas.

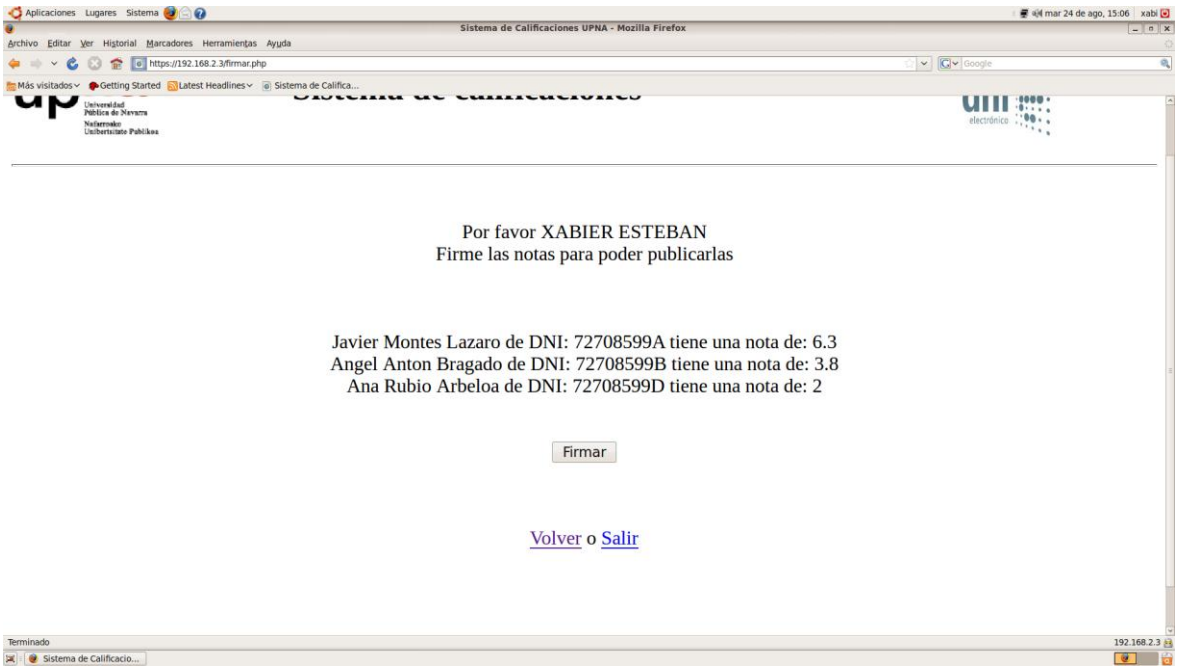


Para una visualización más rápida, la nota de los alumnos aprobados estará en verde y la de los suspendidos en rojo.

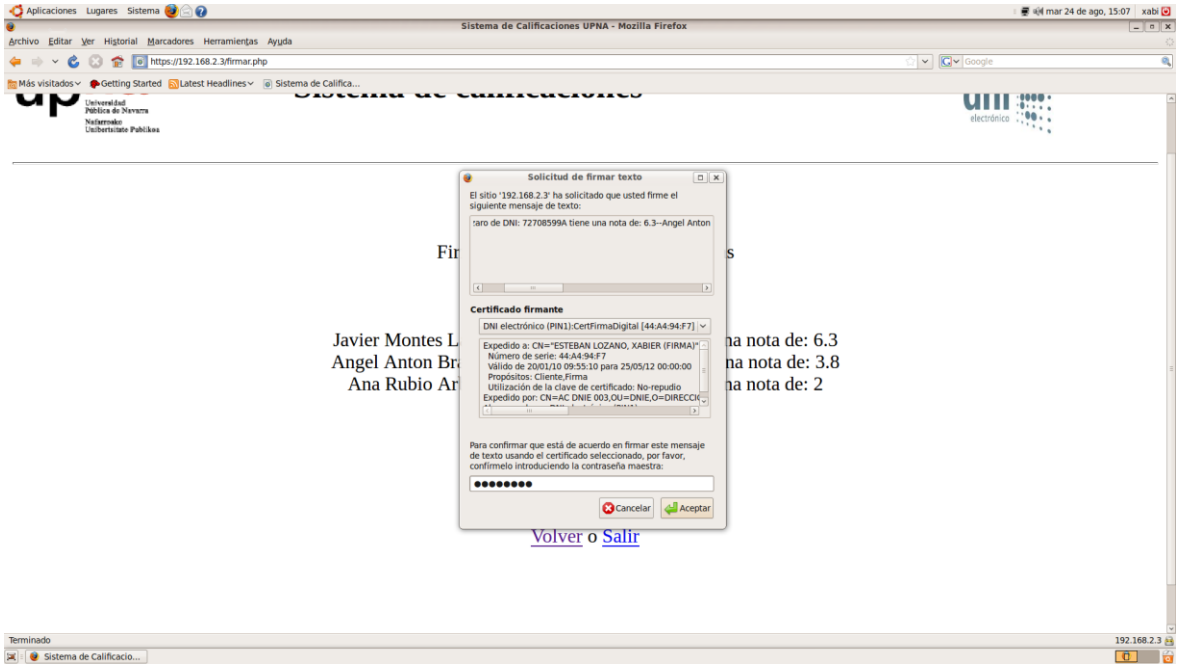
Ahora volvemos al menú, seleccionamos una asignatura y pulsamos sobre publicar notas.



Ahora sólo vemos las notas de una asignatura, pero al lado de la nota podremos seleccionar una nueva nota. Una vez tenemos las notas deseadas, pulsamos en publicar notas.

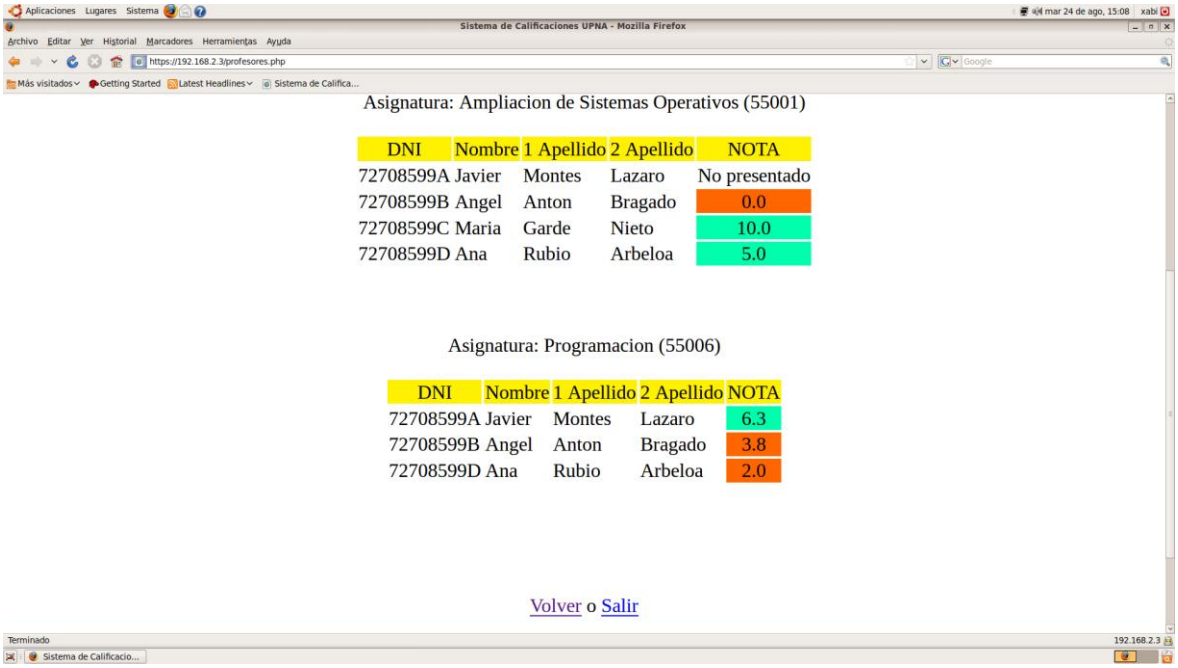


Para poder publicar las notas, es necesario realizar una firma digital de las mismas. Para una mayor seguridad a la hora de firmar, se muestra exactamente lo que se va a firmar.



Una vez pulsamos sobre firmar, el navegador abrirá un menú en el que tenemos que seleccionar el certificado de firma e introducir nuestro código Pin. En el recuadro de arriba se mostrará nuevamente lo que se va a firmar.

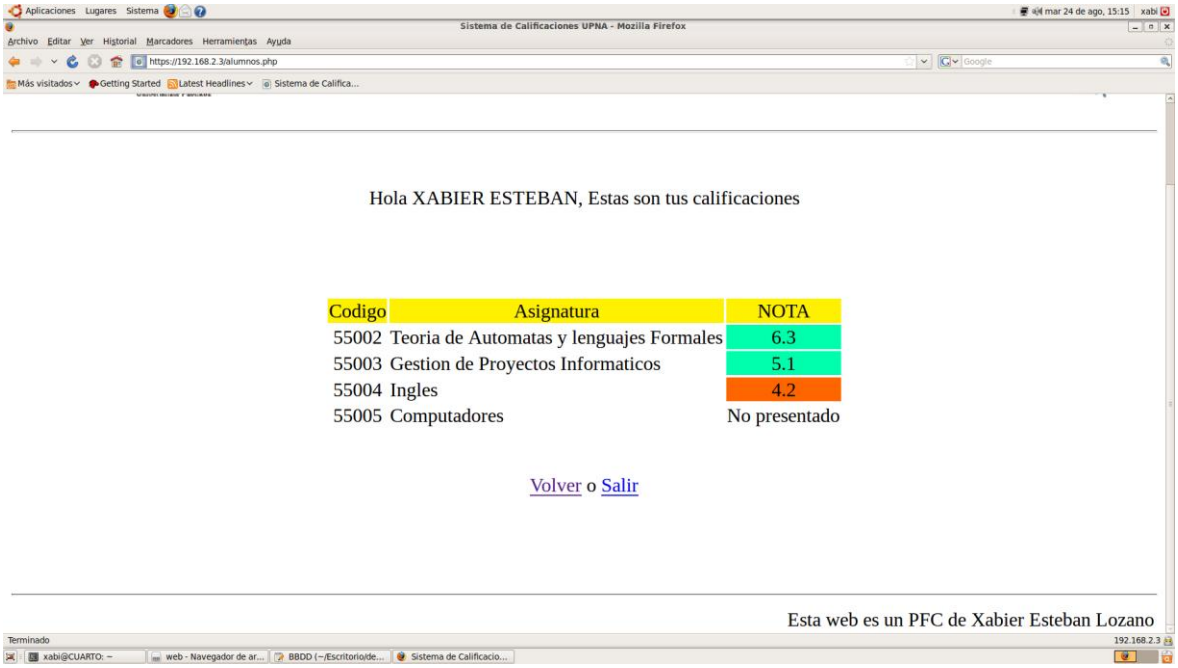
Si todo ha ido bien, obtendremos un mensaje de que la firma ha sido validada correctamente, y que las notas han sido publicadas. Para comprobar que realmente han cambiado volvemos a la opción de consultar notas.



Como podemos comprobar, las notas de la asignatura de programación han sido modificadas.

Si en vez de ser profesores fuésemos alumnos, solamente podríamos consultar nuestras notas, con lo que este sería nuestro menú.

Y si pulsamos sobre ver notas obtendríamos una tabla con todas nuestras calificaciones, de forma muy parecida a la anterior.



4. Conclusiones:

La realización de este proyecto ha servido para comprender mejor cómo funciona el DNIe y algunos de los aspectos más importantes de criptografía. Pese a que el uso de claves y certificados para realizar autentificaciones o firmas digitales no sea algo nuevo, sí que permite a todos los ciudadanos disponer de los métodos para realizarlo de forma segura y gratuita.

El DNIe se considera un dispositivo seguro, por lo que es usado para realizar trámites con la administración pública o incluso con entidades bancarias. Esta seguridad se basa en asegurar el canal, y ciertamente lo consigue, pero en ningún momento se habla de la seguridad de los host. Esto nos obliga a confiar en que la seguridad del servidor no ha sido comprometida, pero asegurar que la seguridad de un ordenador personal no haya sido comprometida a día de hoy se antoja bastante complicado. Aunque esto propiamente no sea un problema de seguridad del DNIe, es importante tenerlo muy en cuenta, debido a la gran confianza que se le otorga al DNIe.

En cuanto a la autentificación, identificar a cada persona de forma segura acabaría con el anonimato en internet, algo que sin duda generará debate, pero sería una gran ventaja desde el punto de vista de la seguridad. Cada vez usamos más internet, para la realización de tares cotidianas, pero seguimos sin saber realmente con quien estamos tratando. De todas formas, ya hemos visto que para realizar una autentificación debemos realizar una consulta OCSP a un servidor, por lo que un ataque a este servidor dejaría sin poder hacer uso del DNIe. Por este motivo no está para nada aconsejado el uso del DNIe para tareas críticas como pueda ser el acceso a un hospital o el control del tráfico. Así pues, aunque podría ser un primer paso para acabar con el anonimato en internet, de momento el DNIe está lejos de este objetivo.

La otra gran ventaja que ofrece el DNIe es la posibilidad de realizar firmas digitales con la misma validez legal que tendría una firma manuscrita. Esto realmente es la gran ventaja que ofrece el DNIe, ya que permite la realización de todo tipo de trámites a través de internet. Esto supone una gran ventaja para el ciudadano ya que los servicios están disponibles a cualquier hora del año y sin necesidad de desplazarse ni hacer colas, lo que repercute en un ahorro de personal, y también un ahorro importante en papel. Una vez dicho esto, sorprende que aunque informáticamente no sea difícil hacerlo, la firma de un correo electrónico no tiene validez legal. Esto es así porque se consideró que la dirección de correo electrónico es un dato demasiado efímero para asegurar que pertenece a una persona en concreto. La solución propuesta es enviar un correo electrónico con un archivo adjunto (en .doc o .pdf por ejemplo) que esté firmado.

Pese a las ventajas que ofrece, el uso del DNIe no está todavía muy extendido, pero si un día llega a estarlo, necesitaríamos asegurar mucho más la seguridad de un ordenador personal, para poder realizar trámites más delicados como puede ser una votación para las elecciones. Así pues, debemos considerar el DNIe como un servicio más para los ciudadanos pero no como una herramienta definitiva de seguridad.

5. Bibliografía:

La forma más rápida y fácil de conseguir información sobre el DNIe es consultar en sus páginas oficiales. Sobre los aspectos básicos de criptografía que necesitaremos para realizar el proyecto, también hay muchísima información en la red. Esta es la bibliografía básica:

- http://www.dnielectronico.es/
- http://www.opensc-project.org/opensc
- http://www.openssl.org/
- http://www.apache.org/
- http://es.wikipedia.org
- https://zonatic.usatudni.es/
- http://www.php.net/

Como bibliografía complementaria:

- http://www.linuxhispano.net/2009/10/28/instalar-mysql-server-ubuntu/
- http://www.linuxtotal.com.mx/index.php?cont=info_seyre_001
- http://blog.osusnet.com/2008/10/11/usando-certificados-ssl-de-cliente-como-sistema-de-autenticacion-web/
- http://oficinavirtual.mityc.es/componentes/MITyCLibXADES/integrator-guide/xades-epes-sign.html
- http://proyectostic.uji.es/pr/cryptoapplet/
- http://doku.locolandia.net/howto/apache-dnie-auth
- http://www.csi.map.es/csi/pg5a12.htm
- http://www.kriptopolis.org/autenticacion-login-con-el-dni-electronico-en-debian-etch-gnu-linux?page=1
- http://infow.wordpress.com/2009/01/11/openssl-la-navaja-suiza-del-cifrado/
- http://doku.locolandia.net/howto/tarjetas-pki
- https://securehomes.esat.kuleuven.be/~decockd/wiki/bin/view.cgi/Using/OpenSC
- http://oasis.dit.upm.es/~jantonio/firmadigital/
- http://spi1.nisu.org/recop/al01/pepe/X509_Kpublica.html
- http://dns.bdat.net/documentos/certificados_digitales/x249.html
- http://foro.colombiaunderground.org/index.php?topic=5280.msg31806;topicseen
- http://foro.portalhacker.net/index.php/topic,96323.0/wap2.html
- http://old.nabble.com/OpenSSL-demos-ssl-not-compiling-td23499380.html